Optimisation de la performance des entrepôts de données XML par fragmentation et répartition

Hadj Mahboubi

Université Lumière Lyon 2
École Doctorale Informatique et Information pour la Société

THÈSE

pour obtenir le grade de
Docteur en Informatique

présentée par

Hadj Mahboubi

Optimisation de la performance des entrepôts de données XML par fragmentation et répartition

Sous la direction du
Pr. Jérôme Darmont

Soutenue publiquement le 8 décembre 2008 devant le jury :

M. Lotfi Lakhal	Université de la Méditerranée (Aix-Marseille II)	(rapporteur)
M. Franck Ravat	Université de Toulouse 1	(rapporteur)
Mlle Angela Bonifati	Consiglio Nazionale delle Ricerche (Icar-CNR)	(examinatrice)
Mme Maryvonne Miquel	INSA Lyon	(examinatrice)
M. Abdelkader Zighed	Université Lumière Lyon 2	(examinateur)
M. Jérôme Darmont	Université Lumière Lyon 2	(directeur de thèse)

Résumé

Les entrepôts de données XML forment une base intéressante pour les applications décisionnelles qui exploitent des données hétérogènes et provenant de sources multiples. Cependant, les Systèmes de Gestion de Bases de Données (SGBD) natifs XML actuels présentent des limites en termes de volume de données gérable, d'une part, et de performance des requêtes d'interrogation complexes, d'autre part. Il apparaît donc nécessaire de concevoir des méthodes pour optimiser ces performances.

Pour atteindre cet objectif, nous proposons dans ce mémoire de pallier conjointement ces limitations par fragmentation puis par répartition sur une grille de données. Pour cela, nous nous sommes intéressés dans un premier temps à la fragmentation des entrepôts des données XML et nous avons proposé des méthodes qui sont à notre connaissance les premières contributions dans ce domaine. Ces méthodes exploitent une charge de requêtes XQuery pour déduire un schéma de fragmentation horizontale dérivée.

Nous avons tout d'abord proposé l'adaptation des techniques les plus efficaces du domaine relationnel aux entrepôts de données XML, puis une méthode de fragmentation originale basée sur la technique de classification k-means. Cette dernière nous a permis de contrôler le nombre de fragments. Nous avons finalement proposé une approche de répartition d'un entrepôt de données XML sur une grille.

Ces propositions nous ont amené à proposer un modèle de référence pour les entrepôts de données XML qui unifie et étend les modèles existants dans la littérature.

Nous avons finalement choisi de valider nos méthodes de manière expérimentale. Pour cela, nous avons conçu et développé un banc d'essais pour les entrepôts de données XML : XWeB. Les résultats expérimentaux que nous avons obtenus montrent que nous avons atteint notre objectif de maîtriser le volume de données XML et le temps de traitement de requêtes décisionnelles complexes. Ils montrent également que notre méthode de fragmentation basée sur les k-means fournit un gain de performance plus élevé que celui obtenu par les méthodes de fragmentation horizontale dérivée classiques, à la fois en terme de gain de performance et de surcharge des algorithmes.

Mots-clefs Données complexes, entrepôts de données XML, fragmentation, grille de données, performance, répartition, SGBD natifs XML, XQuery.

Abstract

XML data warehouses form an interesting basis for decision-support applications that exploit heterogeneous data from multiple sources. However, XML-native database systems currently suffer from limited performances, both in terms of manageable data volume and response time for complex analytical queries. It is therefore necessary to design methods to optimize performances.

In this thesis, we propose to address both these issues by fragmenting and distributing XML data warehouses on grids. To the best of our knowledge, we propose the first fragmentation methods for XML data warehouses. These methods exploit an XQuery workload and output a derived horizontal fragmentation schema.

We first adapted the most efficient fragmentation methods from the relational context to XML, and then proposed an original k-means-based fragmentation method that allows mastering the number of fragments. We finally propose an approach aimed at distributing XML data warehouses on grid architectures.

Our proposals exploit a unified XML warehouse reference model that we propose to synthesize and enhance related work from the literature.

Finally, we experimentally validate our proposal and compare our fragmentation and distribution methods. For this purpose, we designed and developed an XML data warehouse benchmark: XWeB. Our results show that our methods help overcome the data volume and query execution time limitations. They also show that our k-means-based fragmentation method outperforms classical derived horizontal fragmentation methods, both in terms of performance gain and overhead.

Keywords Complex data, data distribution, data grid, fragmentation, performance, XML data warehouses, XML-native DBMSs, XQuery.

Remerciements

J'exprime tout d'abord tous mes remerciements à mon directeur de thèse, M. Jérôme DARMONT, qui m'a donné la chance de suivre cette thèse, qui m'a soutenu, encouragé et offert la possibilité de découvrir le métier d'enseignant chercheur dont j'espère exercer. Je l'en remercie très chaleureusement.

Je tiens aussi à remercier M. Franck RAVAT et M. Lotfi LAKHAL, qui m'ont fait l'honneur de porter intérêt à ma thèse et d'en être les rapporteurs. Je remercie également Mlle Angela BONIFATI, Mme Maryvonne MIQUEL et M. Djamel Abdelkader ZIGHED d'avoir accepter d'être les examinateurs de ce travail.

Je tiens également à exprimer mes remerciements aux membres du laboratoire ERIC, en particulier M. Stéphane LALLICH , M. Jean-Hugues CHAUCHAT et M. Ricco RAKOTOMALALA, pour leurs conseils et leurs orientations. Je voudrais également remercier Mme Valérie Pietroforte Gabriele et Mme Corinne MONTMEAT qui m'ont souvent éclairé certaines démarches administratives.

Toutes mes pensés vont à M. Nicolas NICOLOYANNIS, mon ancien directeur de thèse. Merci Monsieur de m'avoir accueilli et conseillé.

Je souhaite également adresser mes remerciement aux membres de l'équipe BDD, M. Omar BOUSSAÏD, Mme Fadila BENTAYEB, Mme Nouria HARBI et Mme Sabine LOUDCHER RABASEDA. J'ai eu un immense plaisir de travailler avec eux.

J'oublie pas mes collègues du laboratoire, Cécile, Nora, Emna, Anna, Elie, Marouane, Hakim avec qui j'ai partagé des moments de travail inoubliable et à qui je souhaite beaucoup de réussite.

Merci à mes parents, mes sœurs et mon frère qui, malgré l'éloignement, ont cru en moi et m'ont soutenu. Merci à la famille BENKADDOUR de m'avoir hébergé et encouragé tout au long de mes années de thèse.

Merci à tous ceux qui ont participé de près ou de loin à l'aboutissement de ces travaux.

Table des matières

Liste des figures

Liste des tableaux

Notations

Nous utilisons tout au long de ce document les notations suivantes pour les objets que nous manipulons.

d	dimension
a	attribut
f	faits
D	ensemble des dimensions
$\|d\|$	nombre d'instances de la dimension d
A	ensemble d'attributs XML
E	ensemble d'éléments XML
$\mathcal{D}$	ensemble de données
G	graphe XML
P	ensemble de prédicats
p	prédicat
W	charge de requête
q	requête
f_q	fréquence de la requête q
m	nombre de requêtes
F	ensemble de fragments
$frag$	fragment
k	nombre de fragments
Aff	matrice d'affinités
PUM	matrice d'usage des prédicats
$Freq$	Vecteur de fréquences des requêtes
c	cycle
C	ensemble de cycles
$cellsize$	taille physique d'un élément XML

Chapitre 1

Introduction

1.1 Contexte

Actuellement, les données utilisées et échangées par les applications décisionnelles sont de plus en plus en plus diverses et hétérogènes (Darmont & Boussaïd, 2006). En effet, le besoin d'exploiter des données qui ne sont ni numériques ni symboliques émerge dans de nombreux domaines (gestion de la relation client, marketing, veille concurrentielle, médecine...).

Par exemple, l'analyse de données médicales relatives à des patients peut amener à construire un entrepôt de données (définition 1) pour intégrer et exploiter conjointement différentes informations disponibles sous des formes variées : dossiers de patients (base de données classique), antécédents médicaux (textes), radiographies, écographies (documents multimédia), électrocardiogrammes (séries temporelles), diagnostics de médecins (textes ou enregistrements audios), etc. (Darmont, 2006). De telles données sont dites complexes (Darmont et al., 2005b).

Définition 1 *Un entrepôt de données est une collection de données orientées sujet, intégrées, non volatiles, historisées et organisées pour supporter un processus d'aide à la décision (Inmon, 2005). La conception d'un entrepôts de données se base sur deux concepts fondamentaux : les faits (définition 2) et les dimensions (définition 3).*

Définition 2 *Un fait représente un sujet d'analyse et est caractérisé par une ou plusieurs mesures ou indicateurs ayant des propriétés d'additivité.*

Définition 3 *Une dimension est un axe d'observation selon lequel un fait est observé. Une dimension peut comporter plusieurs hiérarchies impliquant différents niveaux de précision dans la description des faits. Ces hiérarchies permettent d'observer des mesures selon plusieurs niveaux de granularité et de construire des agrégats à partir des faits du niveau le plus fin.*

Une réponse possible à la problématique d'entreposage de données complexes s'appuie sur le langage XML (*eXtensible Markup Language* (Bray et al., 2006)). En effet, le langage XML permet de représenter aisément tous types de données peu structurées (Darmont et al., 2005b). Il offre également une grande flexibilité de représentation des données et des possibilités très intéressantes de structuration, de modélisation, de stockage et d'interrogation. XML s'impose également comme un formalisme de représentation des données utilisées dans les processus décisionnels (Beyer et al., 2004; Beyer et al., 2005) et constitue un format de référence pour l'échange de données, notamment de métadonnées représentées par des Définitions de Types de Données (DTD) ou des schémas XML (Hümmer et al., 2003).

Cependant, l'émergence des données XML amène de nouvelles considérations pour les outils et approches d'entreposage et d'analyse, car elles présentent des structures irrégulières et un contenu textuel. Néanmoins, les principes de base de l'entreposage restent valides pour ces données.

Dans cette optique, plusieurs travaux émergent. Les stratégies qu'ils proposent couvrent plus ou moins les différentes étapes du processus d'entreposage de données XML. Comme le montre la figure 1.1, ce processus consiste en trois phases principales.

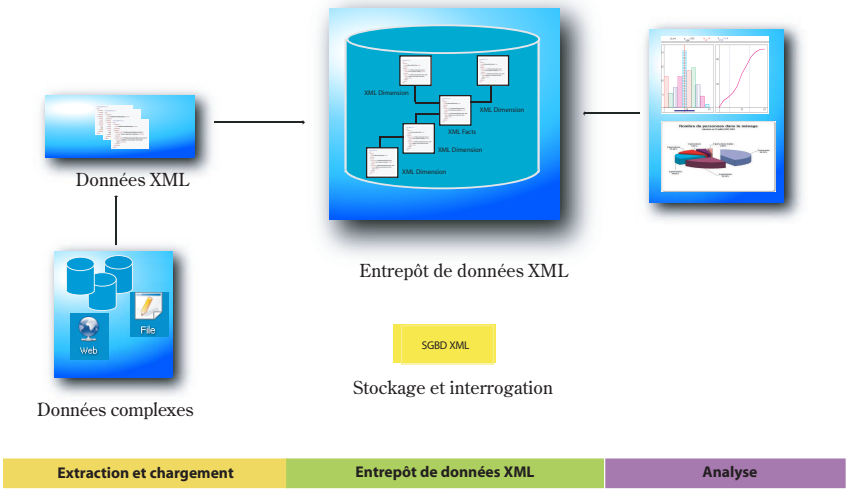

FIG. 1.1 – Processus d'entreposage de données XML

1. *Extraction et chargement des données.* Plusieurs travaux traitent de cette phase. Ils proposent des stratégies d'intégration de sources XML hétérogènes et réparties dans un entrepôt en se basant sur des DTD et des schémas XML (Boussaïd et al., 2006).
2. *Entreposage.* Des travaux récents définissent des modèles d'entrepôts XML (définition 4) spécifiques pour l'analyse des données XML. Ces travaux divergent

souvent dans leurs manières de modéliser l'entrepôt.

3. *Analyse*. Des travaux ont également été entrepris pour l'analyse adéquate et efficace des données d'un entrepôt XML (Hachicha et al., 2007, 2008; Wiwatwattana et al., 2007; Mahboubi et al., 2009a). Ils visent à proposer des solutions et des opérateurs d'analyse de données XML contenant des structures irrégulières. D'autres proposent de plus l'utilisation de techniques de fouille pour une analyse intelligente (BenMessaoud et al., 2006).

Définition 4 *Un entrepôt XML est une collection de données XML organisées de manière multidimensionnelle afin de supporter des requêtes décisionnelles XML (Mahboubi & Darmont, 2009b).*

1.2 Problématique

Les principales caractéristiques des entrepôts de données sont leur grande taille et la complexité des requêtes décisionnelles qu'ils exploitent. Cette complexité est due aux opérations de jointure et d'agrégation utilisées par les requêtes, qui détériorent de manière significative les performances de l'entrepôt lors d'une interrogation portant sur de gros volumes de données. Ces caractéristiques perdurent dans les entrepôts de données XML.

Cependant, comme nous l'avons évoqué précédemment, les travaux dans le contexte de l'entreposage de données XML sont récents et rares sont ceux qui s'intéressent aux performances. De plus, les techniques d'optimisation des performances existantes dans le contexte des bases de données XML s'avèrent inadaptées à la structure particulière des entrepôts XML.

Par ailleurs, les systèmes de gestion de base de données (SGBD) natifs XML sont les plus appropriés pour le stockage et l'interrogation des données des entrepôts XML. En effet, à l'inverse des systèmes relationnels compatibles XML, un SGBD natif XML permet le chargement et le stockage de données XML sans modifier leur structure et offre la possibilité de formuler des requêtes XML portant sur plusieurs documents XML hétérogènes.

Cependant, les SGBD natifs XML actuels présentent des performances insuffisantes quand le volume de données est important et/ou que les requêtes sont complexes. En effet, ces systèmes :

1. stockent les données sous forme d'arbres ou de graphes. Ces structures requièrent un espace mémoire important lors de leur chargement et de leur manipulation ;

2. utilisent des schémas spécifiques (*numbering schemas*) pour la gestion du contenu des éléments et des attributs des données XML. L'utilisation de ces schémas est indispensable pour la récupération des données, ce qui entraîne un coût supplémentaire lors du traitement des requêtes ;
3. sont récents et ne sont pas encore dotés de structures et d'outils d'optimisation de performance (des index, par exemple) ;
4. sont spécialement conçus pour des applications transactionnelles qui exploitent des requêtes simples basées sur des expressions de chemin ;
5. sont ciblés (dans un contexte d'entreposage) par des requêtes décisionnelles qui utilisent des opérations de jointure sur de multiples documents XML. Ces requêtes sont coûteuses car, lors de leur traitement, plusieurs documents XML (graphes) sont simultanément chargés en mémoire, ce qui détériore d'une manière significative les performances du système lorsqu'il sont volumineux (phénomène de swap).

Il apparaît donc nécessaire de concevoir des techniques et outils pour l'optimisation des performances des SGBD natifs XML en général et des entrepôts de données XML en particulier.

1.3 Contributions

Notre travail vise à proposer des techniques d'optimisation de performance pour les entrepôts de données XML et plus particulièrement à traiter conjointement les problématiques du volume des entrepôts XML et de la performance des requêtes décisionnelles. Notre idée consiste à distribuer l'entrepôt pour, d'une part, répartir les données sur plusieurs sites (division de volume) et, d'autre part, permettre un traitement parallèle des requêtes pour l'accélérer. Généralement, un processus de distribution fragmente tout d'abord un ensemble de données avant de le répartir. Nous procédons donc en deux étapes : fragmentation de l'entrepôt XML, puis répartition.

Les techniques de fragmentation existantes dans le contexte des bases de données XML ne sont pas adaptées à nos besoins. Elles proposent en effet de fragmenter un seul document XML pour minimiser le coût de traitement de requêtes simples. De plus, aucune technique de fragmentation n'a été spécialement proposée pour les entrepôts de données XML. En revanche, dans le contexte des entrepôts de données relationnels, la fragmentation a été largement étudiée. Les méthodes proposées exploitent une fragmentation horizontale dérivée qui a été démontrée la plus efficace pour le traitement des requêtes décisionnelles (Bellatreche, 2003). Nous proposons

donc dans un premier temps d'adapter les approches de fragmentation existantes les plus performantes aux entrepôts de données XML (Mahboubi & Darmont, 2009a).

Mais, par ailleurs, ces techniques produisent un nombre important de fragments, inconnu et impossible à prévoir *a priori*. Or, ce nombre doit être connu pour pouvoir distribuer les fragments de l'entrepôt. Afin de le contrôler, nous proposons une technique de fragmentation originale basée sur la technique de classification k-means (Mahboubi & Darmont, 2008).

Nous proposons également une approche de distribution d'entrepôts de données XML sur une grille. Cette approche consiste à répartir les fragments d'un entrepôt XML sur les sites d'une grille qui exploite des modules pour le stockage et l'interrogation.

Les méthodes que nous proposons nécessitent de disposer d'un modèle d'entrepôt de données XML pour pouvoir s'appliquer. Nous proposons donc d'unifier et d'étendre les travaux existants dans le contexte de l'entreposage de données XML (Mahboubi et al., 2009a).

Finalement, la fragmentation horizontale dérivée étant un problème NP-complet résolu par des heuristiques, nous avons choisi de valider nos méthodes de fragmentation de façon expérimentale. Pour réaliser nos expériences, nous avons dû concevoir un banc d'essais pour les entrepôt de données XML : XWeB (Mahboubi & Darmont, 2006).

1.4 Organisation du mémoire

Le reste de ce mémoire est organisé comme suit. Dans le chapitre 2, nous présentons un état de l'art des travaux concernant l'entreposage de données XML, la fragmentation de données dans les contextes relationnel et XML, ainsi que les approches de répartition de données sur une grille, respectivement. Nous présentons ensuite notre modèle de référence pour les entrepôts de données XML dans le chapitre 3. Notre contribution principale est un ensemble de techniques de fragmentation pour les entrepôts XML que nous détaillons dans le chapitre 4. Nous présentons dans le chapitre 5 la démarche que nous adoptons pour la répartition d'un entrepôt de données XML sur grille. Les résultats des expériences que nous avons menées pour valider nos propositions sont présentées dans le chapitre 6. Enfin, le chapitre 7 conclut ce mémoire. Nous y dressons le bilan de nos contributions et nous présentons nos perspectives de recherche.

Chapitre 2

État de l'art

Notre travail traite de la performance des entrepôts de données XML. Afin de mieux aborder les problèmes liés à ce contexte, nous avons dans un premier temps mené une étude bibliographique portant sur les travaux qui traitent de l'entreposage de données XML. Nous discutons et comparons ces travaux afin d'identifier ceux qui sont adaptés à notre contexte (section 2.1). Nous présentons ensuite les techniques de fragmentation dans le contexte des entrepôts relationnels et des bases de données XML, ainsi que celles qui exploitent des techniques de fouille (section 2.2). Finalement, nous présentons les travaux qui traitent de la répartition dans le contexte des bases et entrepôts de données (section 2.3).

2.1 Entreposage de données XML

Les travaux menés dans le contexte de l'entreposage de données XML peuvent être divisés en deux familles (Mahboubi et al., 2009a).

- La première famille propose une modélisation multidimensionnelle pour les entrepôts de données XML. Elle se base sur les modèles classiques (schémas en étoile et dérivés). Ces travaux permettent ainsi une utilisation dynamique des dimensions et offrent un support pour des outils d'analyse.
- Les approches de la seconde famille abordent la problématique de l'entreposage de documents XML. Elles perçoivent un entrepôt XML comme une collection de documents XML sans se baser sur un modèle particulier.

2.1.1 Entrepôts de données XML

Plusieurs travaux traitent de la conception et de la construction d'entrepôts de données XML. Ils se basent sur un schéma en étoile pour modéliser un entrepôt de

données XML et utilisent une collection de documents XML pour représenter les faits et les dimensions. Cet aspect favorise un stockage natif XML et une interrogation via des langages de requêtes XML.

Travaux de Golfareli *et al.* (2001) Dans ces travaux, les auteurs proposent une approche semi-automatique pour la construction de magasins de données (définition 5) depuis des sources XML (Golfarelli et al., 2001). Cette approche manipule directement les données depuis leurs sources et se base sur des schémas XML[1] qui les valident. Elle consiste à identifier un ou plusieurs faits (sujets d'analyse) depuis des schémas XML. Cette identification est effectuée manuellement par le concepteur. Pour chaque fait identifié, un arbre d'attributs est construit. Cet arbre représente les relations entre l'élément fait et le reste des éléments des données XML (dimensions possibles). Il est généré en parcourant les schémas XML qui valident les données XML sources. L'arbre est finalement réarrangé afin de définir les dimensions et les mesures souhaitées.

Définition 5 *Un magasin de données est un extrait d'un entrepôt orienté métier ou activité (on parle de données verticalisées). Un magasin de données permet des analyses plus rapides car portant sur un volume de données plus restreint.*

XML-star schema (2002) Pokorný propose une structure de données nommée XML-star schema (Pokorný, 2002). Un schéma en étoile XML, selon Pokorný, est représenté par un ensemble de documents XML logiquement reliés entre eux. Pour cela, Pokorný introduit la notion de contrainte d'intégrité référentielle dans le contexte XML. Ces documents représentent les faits et les dimensions du schéma. Ils sont extraits des données sources (plusieurs bases de données OLTP -*On-Line Transaction Processing*-) et sont valides selon des sous-DTD extraites des DTD sources.

Travaux de Vrdoljak *et al.* (2003) Les auteurs proposent une méthode pour la construction d'un entrepôt de données XML issues du Web (Vrdoljak et al., 2003). Ils se basent sur des schémas XML qui décrivent et valident les données sources. La méthode consiste à extraire les schémas XML des sources de données, à construire et à transformer ces schémas en graphes de données, à identifier les faits et finalement à construire un schéma XML logique qui valide les données de l'entrepôt. Les auteurs implémentent un prototype qui prend en entrée un schéma XML et produit en sortie un schéma logique de l'entrepôt.

[1] http://xmlfr.org/w3c/TR/xmlschema-0/

XCube (2003) Hümmer *et al.* utilisent des documents XML pour le transfert de cubes de données entre différentes entreprises (Hümmer et al., 2003). Les faits et les dimensions sont définis par un modèle nommé XCube comprenant trois types de documents XML.

1. Le document *XCubeSchema.xml* décrit la structure multidimensionnelle d'un cube de données. Il modélise les dimensions et les mesures. La structure de ce document est décrite dans la figure 2.1 (a). Les lignes discontinues de la figure représentent les relations père-fils et le symbole « * » indique que le nombre d'occurrences d'un fils peut être arbitraire (un ou plusieurs). Le nœud racine *multidimensionalSchema* est composé des nœuds *cubeSchema* et *classSchema*. Le nœud *cubeSchema* décrit la structure multidimensionnelle d'un cube de données et *classSchema* spécifie les dimensions et leurs niveaux hiérarchiques.
2. Le document *XCubeDimension.xml* formalise les dimensions. Il regroupe toutes les dimensions, y compris leurs attributs et leurs valeurs. La structure de ce document est représentée dans la figure 2.1 (b).
3. Le document *XCubeFact.xml* spécifie les faits, c'est-à-dire les identificateurs des dimensions et la description des mesures. La structure de ce document est représentée dans la figure 2.1 (c). Le nœud racine *cubeFacts* est composé d'un ou plusieurs nœuds *cube*. Un nœud *cube* représente un cube de données et est constitué de nœuds *cell* qui décrivent ses faits.

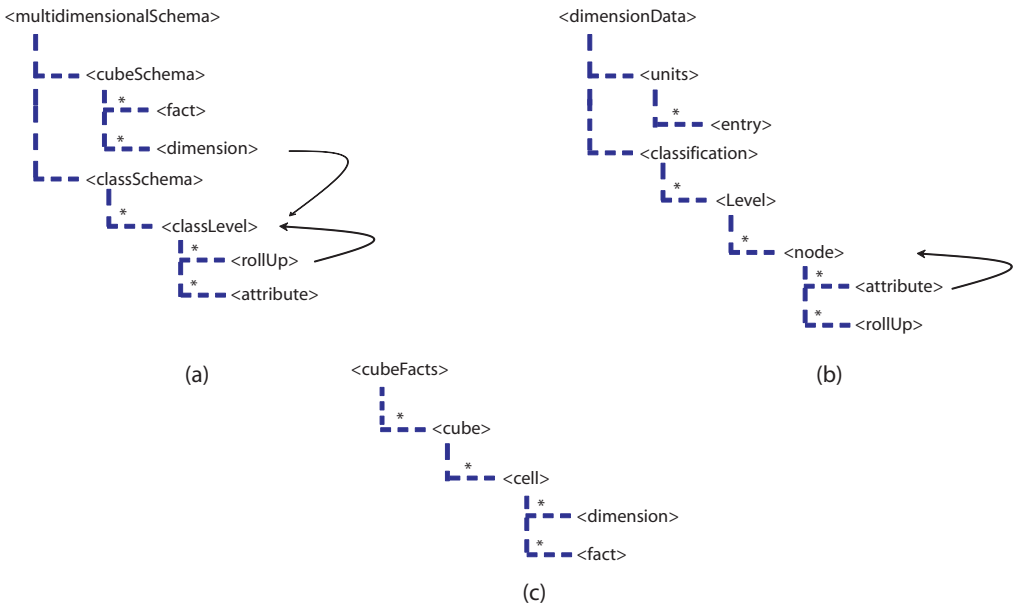

FIG. 2.1 – Structures des documents (a) *XCubeSchema.xml*, (b) *XCubeDimension.xml* et (c) *XCubeFact.xml*

En plus de la description des données du cube, XCube inclut d'autres formats. *XCubeText* fournit des descriptions textuelles et des commentaires sur les schémas et les dimensions. *XCubeQuery* est un moyen d'échange d'informations entre un serveur et un client. Il fournit au site du client les données nécessaires à un besoin défini. Finalement, *XCubeFunction* permet d'interroger un site afin d'extraire les descriptions des cubes de données.

XML-OLAP (2005) Park *et al.* proposent une plateforme pour l'analyse en ligne de documents XML nommée XML-OLAP (Park et al., 2005). Comme le montre la figure 2.2, les auteurs se basent sur un entrepôt de données XML où les faits sont représentés par une collection de documents XML et chaque dimension par une autre collection de documents XML. Les auteurs représentent chaque fait par un document XML. Cette représentation est inspirée de celle adoptée par Nassis *et al.* (Nassis et al., 2005). Un document XML d'une collection de dimensions stocke une instance de la hiérarchie d'une dimension. Les auteurs affirment que cela permet d'éliminer les opérations de jointure entre les niveaux hiérarchiques d'une dimension.

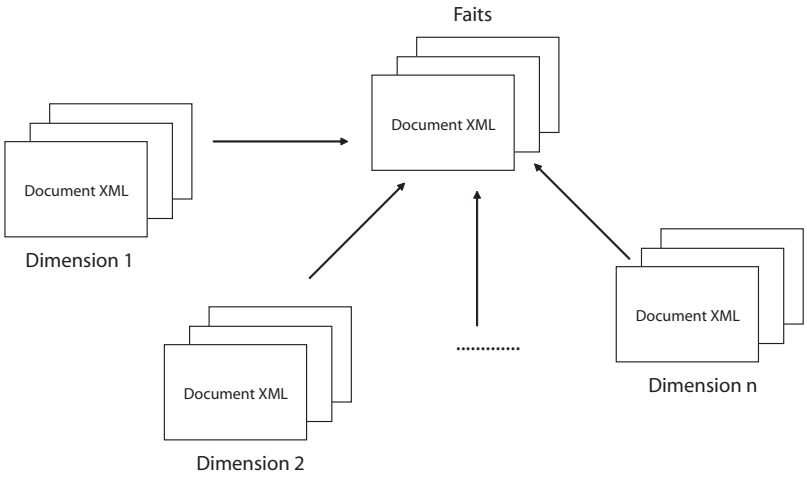

FIG. 2.2 – Modèle multidimensionnel d'entrepôt XML-OLAP

Afin d'interroger l'entrepôt de données XML, les auteurs proposent un langage d'expression multidimensionel : XML-MDX (Park et al., 2005). Ils étendent en fait le langage relationnel MDX de Microsoft avec deux opérateurs : `CREATE XQ-CUBE` pour la création de cubes XML et `SELECT` pour leur interrogation. De plus, les auteurs définissent sept opérateurs d'agrégation : `ADD`, `LIST`, `COUNT`, `SUMMARY`, `TOPIC`, `TOP KEYWORD` et `CLUSTER`. Certains de ces opérateurs sont inspirés du relationnel, d'autres utilisent des techniques de fouille de données textuelles pour agréger des valeurs non-additives.

Travaux de Rusu *et al.* (2005) Les auteurs se basent sur le langage XQuery[2] pour construire un entrepôt de données XML (Rusu et al., 2005). La méthode proposée couvre les processus d'extraction, de transformation, d'agrégation et d'intégration des données XML dans l'entrepôt. Toutes ces étapes sont basées sur le langage XQuery et visent à minimiser la duplication et les erreurs dans les données de l'entrepôt.

Les auteurs tentent également de régler les problèmes de conflits rencontrés lors de l'intégration des données dans l'entrepôt en proposant des règles d'intégration. Ces conflits sont de deux types : conflits dans le schéma, qui surviennent quand le schéma existe mais que les données ne correspondent pas totalement à ses spécifications ; et conflits dans les données, qui correspondent aux irrégularités structurelles des données XML.

Les auteurs stockent les faits et les dimensions dans des documents XML construits par des requêtes XQuery. Par exemple, la figure 2.1.1 présente une requête XQuery qui construit une dimension temps. Finalement, les auteurs affirment que leur approche est générique et peut être appliquée à tout modèle d'entrepôt de données XML.

```
let $b:=0 document {
for $t in distinct-values(doc("libraryBooks.xml")//borrowing_date)
let $b:=$b+1
return <borrowtime>
<timekey>$b</timekey>
<borrowdate>$t</borrowdate>
<month>get-month-from-date($t)</month>
</borrowtime> }
```

FIG. 2.3 – Exemple de requête XQuery de création de dimension

X-Warehousing (2006) Boussaïd *et al.* définissent un entrepôt par un schéma XML. Les auteurs implémentent un prototype, nommé X-Warehousing, pour l'entreposage des données complexes (Boussaïd et al., 2006). La figure 2.4 résume les différentes étapes de l'approche X-Warehousing. Les besoins d'analyse des utilisateurs ainsi que les documents XML sources sont représentés par des schémas XML. Ces schémas sont transformés en arbres d'attributs (Golfarelli et al., 2001). La méthode applique ensuite des fonctions de fusion par élagage (*pruning*) et/ou par greffe (*grafting*) (Golfarelli & Rizzi, 2001) sur ces arbres pour construire un arbre d'attributs final qui représente le schéma de l'entrepôt.

[2]http://www.w3.org/TR/xquery/

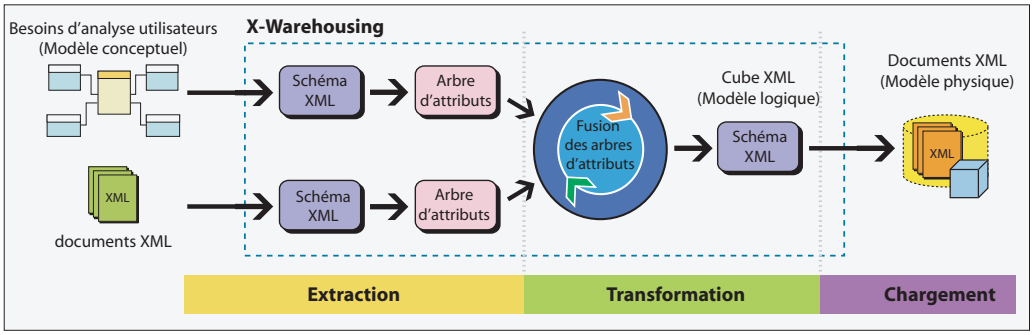

FIG. 2.4 – L'approche X-Warehousing

Dans un entrepôt X-Warehousing, chaque fait, ainsi que les dimensions correspondantes, sont stockés dans un document XML. L'ensemble de documents XML obtenu constitue un cube de données XML. Les auteurs définissent cet ensemble comme un contexte d'analyse.

Synthèse Les travaux qui proposent de construire un entrepôt de données XML sont résumés dans le tableau 2.1. Dans ce tableau, un « + » indique que l'approche utilise ou traite un critère du tableau. Ces travaux utilisent des documents XML pour représenter les faits et les dimensions.

Certains ont pour vocation d'intégrer et d'analyser des données XML exploitées par le commerce électronique et échangées sur le Web. Leur objectif principal est de centraliser les données XML et de les intégrer dans un « entrepôt Web » (Golfarelli & Rizzi, 2001; Vrdoljak et al., 2003). D'autres s'intéressent à l'entreposage de données hétérogènes représentées par des documents XML (Pokorný, 2002; Park et al., 2005; Rusu et al., 2005). Finalement, Hümmer *et al.* proposent de représenter les cubes de données échangés entre entreprises par des documents XML.

Dans ce contexte, certains travaux définissent également des modèles d'entrepôt de données XML qui diffèrent dans la structure des documents XML utilisés pour représenter les faits et les dimensions. Golfarelli *et al.* et Boussaïd *et al.* proposent, par exemple, de représenter chaque fait et les instances des dimensions correspondantes par un seul documents XML, tandis que les autres travaux représentent les faits et les dimensions par des documents XML séparés (Pokorný, 2002; Park et al., 2005; Rusu et al., 2005).

2.1.2 Entrepôts de documents XML

Les travaux qui traitent de l'entreposage de documents XML considèrent un entrepôt comme une collection de documents XML.

Travaux	Modèle multidimensionnel	Données	Représentation des faits et des dimensions
Golfarelli *et al.* (2001)		Web	
Pokorný (2002)	+	Hétérogènes	+
Vrdoljak *et al.* (2003)		Web	
Hümmer *et al.* (2003)	+	Cubes	+
Rusu *et al.* (2005)	+	Hétérogènes	+
Park *et al.* (2005)	+	Hétérogènes	+
Boussaïd *et al.* (2006)	+	Hétérogènes	

TAB. 2.1 – Comparaison des entrepôts de données XML

.

Xyleme (2001) Xyleme est une architecture pour l'entreposage dynamique de données XML collectées sur le Web (Xyleme, 2001). Xyleme stocke les données de l'entrepôt dans le système NATIX qui exploite conjointement une approche relationnelle et une approche native XML pour le stockage des données XML. Xyleme inclut un moteur de traitement de requêtes spécifique aux données XML du Web, qui se base sur une algèbre qui définit de nouveaux opérateurs pour le parcours et l'extraction de données depuis une grande collection de données XML. Le prototype Xyleme exploite aussi des mécanismes de traitement parallèle basés sur des services Web. De plus, il propose une intégration et une interrogation sémantique de données XML de DTD[3] hétérogènes. Il est actuellement exploité par l'entreprise Xyleme[4].

DAWAX (2003) Baril et Bellahsène perçoivent un entrepôt de données comme une collection de vues XML matérialisées (Baril & Bellahsène, 2003). Les vues XML constituent l'entrepôt et permettent de filtrer et de structurer les données pour un besoin d'analyse. Selon cette approche, Baril et Bellahsène ont développé un système appelé DAWAX (*DAta Warehouse for XML*). DAWAX se base sur trois modules principaux.

1. *Module de spécification de l'entrepôt de données.* Ce module définit les vues XML qui constituent l'entrepôt de données. Deux sortes de vues peuvent être distinguées : des vues de sélection et des vues composées qui créent de nouveaux éléments et attributs à partir de plusieurs sources. Des fonctions d'agrégation sont utilisées pour définir les nouvelles valeurs.
2. *Module de gestion des métadonnées.* La spécification de l'entrepôt de données

[3]http://www.w3schools.com/DTD/default.asp
[4]http://www.xyleme.com/

est stockée dans un document XML. Ce document contient des métadonnées qui définissent des informations sur le stockage des données, la provenance (URL) des sources des données et les spécifications des vues. Afin de fournir une intégration de vues sources hétérogènes, l'entrepôt de données est considéré comme un document XML union de toutes les vues. Un élément entrepôt est défini par la ligne de DTD suivante : <!ELEMENT datawarehouse (view1, view2, . . . , viewN)>.

3. *Module de stockage et de gestion des données.* Le stockage des données s'effectue grâce à un *mapping* vers une base de données relationnelle. Il en est de même pour les vues. Pour cela, trois tables sont utilisées : *Patterns* pour le stockage des modèles, *Fragments* pour le stockage des fragments et *Views* pour le stockage des vues.

xFact et GxFact (2005) Nassis *et al.* proposent une approche conceptuelle, nommée xFact, pour concevoir et construire un magasin de données XML (Nassis et al., 2005). Les auteurs exploitent des concepts orientés objets et proposent de sélectionner les dimensions en se basant sur les besoins des utilisateurs. Les dimensions sont représentées par des vues virtuelles (*VDim*), ce qui permet d'enrichir la sémantique de l'entrepôt. Dans cette approche, les auteurs supposent que toutes les dimensions font partie des données faits et que chaque fait est décrit par un seul document XML.

Dans le même contexte, Rajugan *et al.* proposent une approche basée sur les vues pour la modélisation et la construction d'un entrepôt de documents XML, appelée GxFact (Rajugan et al., 2005). GxFact réunit des xFact (Nassis et al., 2005) (magasins et entrepôts XML distribués) pour une exploitation globale. Les auteurs proposent aussi trois stratégies pour la construction et la gestion des GxFacts. L'une propose une matérialisation des dimensions (*fully persistent*), l'autre privilégie des dimensions non matérialisées (*non-persistent*) et la dernière utilise une combinaison des deux autres. Ces stratégies offrent le moyen de modéliser de façons différentes les dimensions *VDim* et leurs hiérarchies (matérialisation ou pas).

X-Warehouse (2005) Zhang *et al.* proposent une approche basée sur les informations extraites des requêtes fréquentes pour matérialiser un entrepôt de documents XML (Zhang et al., 2005). Les auteurs appliquent une classification hiérarchique pour fusionner les requêtes et ainsi construire l'entrepôt.

La méthode proposée consiste à identifier les sources de données fréquemment interrogées par les utilisateurs, ainsi que les requêtes utilisées. Elle transforme ensuite ces requêtes en chemins de requêtes (*Query Path Transactions*) et applique

une technique de fouille pour calculer les chemins les plus fréquents (*Frequent Query Path*). Ces derniers constituent une base pour la construction du schéma de l'entrepôt. Les auteurs prévoient aussi un nettoyage des données avant leur intégration dans l'entrepôt, sans préciser comment le faire.

2.1.3 Synthèse générale

Les travaux qui traitent de l'entreposage de données XML varient par leur approche de construction de l'entrepôt et couvrent plus ou moins les différentes étapes du processus d'entreposage : ETL (Extraction, Transformation et Chargement), modélisation et analyse. Certains proposent de traiter les données directement à partir de leurs sources, d'autres favorisent le chargement dans un entrepôt de données modélisé par un schéma en étoile. Certains proposent également des opérateurs d'analyse spécifiques aux données XML. Ces travaux sont résumés dans le tableau 2.2. Le symbole « + » y indique qu'une approche traite une étape du processus d'entreposage.

Les approches qui proposent de construire des entrepôts de documents XML sont basées sur des vues définies par l'utilisateur. Elles proposent des méthodes de conception et de construction d'un espace de stockage XML (*XML repository*) qui représente le contexte d'analyse de l'entrepôt et ne définissent aucun modèle logique d'entrepôt XML. Ces approches sont orientées besoins et sont plutôt employées lorsque les spécifications de l'entrepôt sont peu susceptibles d'évoluer.

Les approches qui proposent de construire des entrepôts de données Web exploitent des schémas XML et des DTD à des fins de modélisation. Ces formalismes permettent de décrire la structure des données et de définir des relations entre les documents XML sources, qui sont souvent hétérogènes. Ces DTD/schémas sont transformés et fusionnés pour construire le schéma de l'entrepôt (Golfarelli et al., 2001). Cependant, les mécanismes de transformation peuvent s'avérer complexes. Ces approches sont donc utilisées quand les dimensions sont statiques, c'est-à-dire que les besoins d'analyse sont fixés. De plus, ces besoins peuvent parfois ne pas être totalement satisfaits. Ce problème survient notamment quand les données ciblées ne sont pas disponibles. Ces travaux proposent une préparation de données à des fins d'analyse, mais n'incluent aucun outil ni opérateur d'analyse spécifique aux données XML.

Les approches de la famille des entrepôts de données XML respectent une modélisation multidimensionnelle pour construire un entrepôt ou un magasin de données. Ces approches diffèrent principalement dans la manière de représenter les faits et les dimensions, ainsi que par le nombre de documents XML utilisés pour les stocker.

Certains travaux se basent sur des critères de performance pour définir ces éléments. D'autres proposent des modèles simplifiés pour la modélisation de l'entrepôt ou respectent strictement une modélisation en étoile.

Ces approches proposent également des mécanismes d'extraction et de chargement des données dans l'entrepôt. Par exemple, Rusu *et al.* définissent des règles pour le nettoyage et le chargement des données et Park *et al.* s'inspirent des approches orientées objets pour la modélisation et l'intégration des données dans l'entrepôt. Ces approches utilisent des schémas UML construits à partir des DTD des données sources. De plus, Hümmer *et al.* proposent de représenter les métadonnées de l'entrepôt par un document XML, tandis que Pokorný valide les documents XML de l'entrepôt par des DTD. Ces métadonnées n'existent dans aucun autre travail. Or, ce type de document peut s'avérer utile pour l'interrogation et la mise à jour des données de l'entrepôt. Finalement, d'autres travaux proposent des solutions pour l'interrogation des données XML. Par exemple, Hümmer *et al.* définissent un modèle pour l'interrogation de cubes de données XML et Park *et al.* proposent des opérateurs d'analyse adaptés aux données XML.

Les travaux qui construisent des entrepôts de données XML sont donc les mieux adaptés dans notre contexte. En effet, ils proposent des modèles d'entrepôts XML et avancent des outils d'analyse spécifiques aux données XML.

Familles	**Travaux**	**ETL**	**Entrepôt**	**Analyse**
Entrepôts de documents XML	Xyleme (2001)	+		
	Baril et Bellahsène (2003)	+	+	
	Nassis *et al.* (2005)	+		
	Rajugan *et al.* (2005)	+		
	Zhang *et al.* (2005)	+		
Entrepôts de données XML	Golfarelli *et al.* (2001)	+	+	
	Pokorný (2002)	+	+	+
	Vrdoljak *et al.* (2003)	+	+	
	Hümmer *et al.* (2003)		+	+
	Rusu *et al.* (2005)	+	+	
	Park *et al.* (2005)	+	+	+
	Boussaïd *et al.* (2006)	+	+	

TAB. 2.2 – Comparaison des travaux d'entreposage de données XML

.

2.2 Fragmentation de données

La fragmentation (définition 6) est une technique d'optimisation de performance qui a été largement étudiée dans le domaine des bases et entrepôts de données (Bellatreche & Boukhalfa, 2005; Datta et al., 1999; Wu & Buchmann, 1997). Dans cette section, nous présentons et discutons les différentes contributions qui traitent de la fragmentation dans les entrepôts de données relationnels et les bases de données XML. Nous présentons également quelques approches qui exploitent des techniques de fouille de données pour la fragmentation dans les contextes relationnel et orienté objets.

Définition 6 *La fragmentation consiste à diviser un ensemble de données en plusieurs partitions, appelées fragments, de manière à ce que la combinaison des fragments recouvre l'intégralité des données sources sans ajout ni perte d'information (Ma et al., 2006).*

Dans le contexte relationnel, ce sont les tables (relations) qui sont partitionnées ; dans le contexte XML, ce sont les documents XML. La fragmentation peut prendre place dans plusieurs scénarios : la conception de systèmes distribués, le traitement de flux de données et l'échange de données dans les systèmes pair-à-pair. Le résultat d'une fragmentation est appelé schéma de fragmentation (Bellatreche, 2000).

Dans la littérature, trois types de fragmentation sont définis (Bremer & Gertz, 2003; Koreichi & Cun, 1997) : la fragmentation verticale, la fragmentation horizontale et la fragmentation mixte.

- La fragmentation verticale divise un ensemble de données $\mathcal{D}$ en sous-ensembles $\mathcal{D}_i$ (i=*1..n* avec n le nombre de fragments) obtenus par des opérations de projection selon un ou plusieurs attributs a_j (j=*1..m* avec m le nombre d'attributs). La fragmentation verticale consiste à regrouper les attributs fréquemment utilisés par les requêtes. Un fragment vertical est construit par une opération de projection. La relation source est reconstruite par jointure des fragments.
- La fragmentation horizontale partitionne un ensemble de données $\mathcal{D}$ en sous-ensembles $\mathcal{D}_i$ selon des prédicats de sélections p_k (k=*1..r* avec r le nombre de prédicats de sélection). Un fragment horizontal est obtenu par une opération de restriction sur $\mathcal{D}$. La reconstruction de la relation source $\mathcal{D}$ à partir des fragments horizontaux est obtenue par une opération d'union.
  La fragmentation horizontale se décline en deux variantes (Bellatreche, 2000) : la fragmentation primaire et la fragmentation dérivée.

- La fragmentation horizontale primaire est effectuée grace à des prédicats de sélection définis sur l'ensemble de données $\mathcal{D}$. Elle réduit le coût de traitement d'une requête en minimisant le nombre d'accès aux enregistrements non nécessaires.
- La fragmentation horizontale dérivée consiste à partitionner un ensemble de données $\mathcal{D}$ selon des prédicats définis sur un autre ensemble de données $\mathcal{D}'$. La fragmentation horizontale dérivée réduit le coût de traitement d'une opération de jointure entre $\mathcal{D}$ et $\mathcal{D}'$ (Bellatreche, 2000).

– La fragmentation mixte partitionne verticalement des fragments horizontaux ou horizontalement des fragments verticaux. Les algorithmes de fragmentation mixte ont été étudiés dans le contexte relationnel et sont subdivisés en deux types : la fragmentation par création de grille (Navathe et al., 1995) et la fragmentation par définition de vues (Pernul et al., 1991).

Finalement, un schéma de fragmentation doit présenter les trois propriétés suivantes (Bellatreche, 2000).

– La complétude assure que chaque instance de l'ensemble fragmenté appartient à au moins un fragment.
– La reconstruction garantit la reconstitution de l'ensemble de données sources sans perte ni ajout d'information.
– La disjonction indique que les fragments doivent être disjoints deux à deux.

2.2.1 Fragmentation dans les entrepôts de données relationnels

Les travaux qui traitent de la fragmentation dans les entrepôts de données relationnels s'inspirent de ceux proposés dans les bases de données relationnelles (Navathe et al., 1995; Zhang & Orlowska, 1994) et orientées objets (Bellatreche et al., 1998; Ezeife & Barker, 1995; Ravat & Zurfluh, 1995). Cette fragmentation peut être horizontale, verticale ou mixte.

Travaux de Wu et Buchmaan (1997) Les auteurs recommandent de combiner la fragmentation horizontale et la fragmentation verticale (Wu & Buchmann, 1997). Selon eux, la table de faits peut être partitionnée horizontalement à partir de fragments définis sur les dimensions. Elle peut aussi être partitionnée verticalement selon les clés étrangères faisant référence aux dimensions. Cependant, aucun algorithme de fragmentation n'est proposé.

Travaux de Datta *et al.* (1999) Les auteurs exploitent la fragmentation verticale pour construire un index nommé *Cuio* dans un entrepôt modélisé par un schéma en étoile (Datta et al., 1999). *Cuio* permet d'accélérer l'accès aux données et optimise l'espace de stockage en matérialisant les fragments au lieu des attributs indexés.

Travaux de Golfarelli *et al.* (1999) Afin de minimiser le temps de réponse des requêtes, Golfarelli *et al.* utilisent la fragmentation verticale pour partitioner des vues définies sur un entrepôt (Golfarelli et al., 1999). Cette fragmentation est basée sur une charge de requêtes et un modèle de coût. Selon les auteurs, la fragmentation verticale désigne deux opérations : d'une part le partitionnement d'une vue en plusieurs fragments et, d'autre part, l'unification en une seule vue de deux ou plusieurs vues ayant une clé commune. L'unification respecte la règle de reconstruction d'une table fragmentée à partir de ses fragments verticaux (Özsu & Valduriez, 1999) et vise à réduire la redondance des vues. Les auteurs supposent que leur approche peut être bénéfique pour la distribution de l'entrepôt sur une architecture parallèle et proposent de combiner leur algorithme de fragmentation avec un algorithme d'allocation des fragments sur les nœuds distants (Özsu & Valduriez, 1999).

Travaux de Munneke *et al.* (1999) Les auteurs appliquent la fragmentation dans une base de données multidimensionnelle (Munneke et al., 1999) en partitionnant un cube de données en sous-cubes appelés *fragment-cubes*.

Pour cela, les auteurs se basent sur les opérations *slice* et *dice*. Cette fragmentation est similaire à une fragmentation horizontale pour une base de données relationnelle. Elle consiste à partitionner un cube sans modifier sa structure et se base sur des opérateurs de restriction définis sur une ou plusieurs dimensions du cube. La reconstruction du cube original est effectuée par des opérations de jointure.

Munneke *et al.* proposent également un autre type de fragmentation, appelée *server*, qui est équivalente à une fragmentation verticale dans une base de données relationnelle. La fragmentation *server* élimine une ou plusieurs dimensions dans un cube pour produire un fragment. Afin d'assurer la reconstruction des fragments, une ou plusieurs dimensions sont dupliquées dans tous les fragments.

Travaux de Noaman & Barker (1999) Afin de construire un entrepôt de données distribué, les auteurs exploitent une stratégie descendante par fragmentation horizontale (Noaman & Barker, 1999). Elle part du schéma conceptuel global d'un entrepôt, qu'elle répartit pour construire les schémas conceptuels locaux. Cette répartition se fait en deux étapes essentielles : la fragmentation et l'allocation, suivies éventuellement d'une optimisation locale.

Les auteurs proposent un algorithme qui dérive des fragments faits en se basant sur des requêtes définies sur les dimensions.

Travaux de Bellatreche (2000) Bellatreche applique la fragmentation horizontale dérivée sur un schéma en étoile et propose plusieurs approches basées sur un ensemble de requêtes. L'auteur adapte les algorithmes proposés dans le contexte des bases de données réparties. Ces algorithmes se basent sur la complétude et la minimalité des prédicats ou sur les affinités des requêtes (Bellatreche, 2000).

Bellatreche constate que ces méthodes génèrent un nombre important de fragments et rendent ainsi leur processus de maintenance très coûteux. Pour répondre à cette problématique, il propose des algorithmes de sélection d'un schéma de fragmentation optimal. Ces algorithmes visent à trouver un compromis entre le coût de maintenance des fragments et le coût d'exécution des requêtes. Ils sont basés sur des modèles de coût et procèdent en trois étapes : génération de plusieurs schémas de fragmentation, évaluation de ces schémas et sélection d'un schéma optimal.

Le premier algorithme est exhaustif et consiste à construire tous les schémas de fragmentation possibles par fragmentation horizontale. Il énumère ensuite ces schémas et calcule pour chacun d'eux le coût d'exécution des requêtes de la charge. Il sélectionne finalement le schéma qui correspond au coût minimum. Le deuxième algorithme est approximatif. Il construit un schéma initial par l'algorithme de fragmentation dirigée par les affinités, puis l'améliore par des opérations de fusion ou de décomposition des fragments. Finalement, le troisième algorithme exploite un algorithme génétique pour sélectionner un schéma de fragmentation.

Méthodes de fragmentation horizontale La plupart des travaux que nous venons de présenter exploitent la fragmentation horizontale dérivée. Cette fragmentation favorise le traitement de requêtes comprenant de multiples opérations de jointure sur plusieurs relations. Elle empêche aussi l'accès à des données non nécessaires pour le traitement de ces requêtes (Noaman & Barker, 1999; Bellatreche & Boukhalfa, 2005; Wehrle et al., 2005). Dans la littérature, deux méthodes sont principalement utilisées pour la fragmentation primaire. L'une est basée sur la construction de prédicats et l'autre sur les affinités des requêtes (Koreichi & Cun, 1997).

– *Construction de prédicats.* Cette méthode partitionne une relation grâce à un ensemble complet et minimal de prédicats construit par l'algorithme COM-MIN (Noaman & Barker, 1999). La complétude signifie que deux instances d'une relation dans un même fragment ont la même probabilité d'être accédés séparément. La minimalité garantit qu'il n'existe pas de redondance dans les

prédicats. Cet algorithme prend en entrée un ensemble de prédicats de sélection P. Il procède en deux étapes. Dans la première étape, il sélectionne un prédicat complet de P et l'affecte à un ensemble P'. Dans la deuxième étape, il procède de manière itérative. À chaque itération, il affecte à P' un prédicat complet de P pertinent (définition 7) par rapport aux prédicats présents dans l'ensemble P'. P' est l'ensemble complet et minimal de prédicats. L'algorithme s'arrête quand P est vide.

– *Affinités entre requêtes.* Cette méthode est une adaptation de la fragmentation verticale (Navathe et al., 1995). Elle est basée sur le concept d'affinité (Zhang & Orlowska, 1994), qui désigne la fréquence des requêtes. La méthode exploite des matrices spécifiques (matrice d'usage et matrice des affinités) pour regrouper les prédicats de sélection selon les fréquences des requêtes qui les utilisent. Un groupe est identifié par un cycle (ensemble de prédicats) et désigne un fragment de dimension.

Définition 7 *Soient P un ensemble de prédicats de sélection et p un prédicat de sélection. p est pertinent par rapport à l'ensemble P s'il existe un prédicat $p' \in P$ tel que les fragments horizontaux définis par $(p \wedge p')$ et $(p \wedge \neg p')$ sont accédés individuelement par au moins une requête.*

Synthèse Les travaux que nous venons de présenter sont résumés dans le tableau 2.3. Ces travaux proposent des techniques de fragmentation qui interviennent dans deux contextes : centralisé et distribué. Dans un contexte centralisé, ils visent à améliorer le traitement de requêtes décisionnelles. Dans le contexte distribué, ils partitionnent un entrepôt pour le répartir sur des sites géographiquement distants (Noaman & Barker, 1999). Ce processus favorise le traitement parallèle des requêtes. Ces travaux proposent également d'appliquer une fragmentation sur un entrepôt (Bellatreche, 2000), des vues matérialisées (Golfarelli et al., 1999) ou des cubes de données (Munneke et al., 1999).

Travaux	**Contexte**	**Fragmentation**
Wu and Buchmaan (1997)	Centralisé	Entrepôt
Golfarelli *et al.* (1999)	Centralisé	Vues matérialisées
Datta *et al.* (1999)	Centralisé	Entrepôt
Noaman *et al.* (1999)	Distribué	Entrepôt
Munneke *et al.* (1999)	Distribué	Cubes
Bellatreche and Boukhalfa (2005)	Centralisé	Entrepôt

TAB. 2.3 – Comparaison des approches de fragmentation des entrepôts de données

2.2.2 Fragmentation de données XML

Récemment, plusieurs techniques de fragmentation ont été proposées dans le contexte XML. Elles fragmentent un document XML en un nouvel ensemble de documents. L'objectif principal de ces techniques est d'améliorer les performances des requêtes XML (contexte centralisé) (Bonifati & Cuzzocrea, 2007; Gertz & Bremer, 2003; Ma et al., 2003) ou de distribuer ou échanger des données XML dans un réseau (contexte distribué) (Bonifati et al., 2004; Bose & Fegaras, 2005).

Fragmentation XML dans le contexte centralisé Afin de fragmenter un document XML, Ma *et al.* définissent un nouveau type de fragmentation nommé *split* (Ma & Schewe, 2003; Ma et al., 2003). Cette fragmentation est inspirée de celle utilisée dans les bases de données orientées objets (Schewe, 2002). Elle consiste à partitionner un élément du document XML en sous-éléments et à leur attribuer des références pour les relier. Ces références sont aussi ajoutées à la DTD qui spécifie le document XML source. Pour cela, les auteurs définissent une extension de la DTD classique. Dans la nouvelle DTD, les attributs sont définis séparément par des balises ATTRIBUTE et non par une liste de déclarations ATTLIST. Cette DTD spécifie aussi d'autres types de données (INT, STRING...) en plus des types CDATA et PCDATA habituels et comporte des clefs (ID) qui permettent de référencer d'autres documents XML.

Bonifati *et al.* traitent la problématique de la limitation d'espace mémoire pour le traitement de requêtes XML (Bonifati & Cuzzocrea, 2007; Bonifati et al., 2006). Pour cela, les auteurs utilisent une fragmentation dirigée par les contraintes structurelles du document XML (taille, largeur et profondeur). La méthode proposée parcourt les nœuds du document XML de manière descendante et applique à chaque nœud des heuristiques qui exploitent les contraintes structurelles du document et permettent de déterminer si un nœud constitue la racine d'un fragment.

Andrade *et al.* proposent la fragmentation d'une collection XML (ensemble de documents XML homogènes) et non d'un seul document XML (Andrade et al., 2006). Cette fragmentation est basée sur l'algèbre TLC (*Tree Logical Class*) (Paparizos et al., 2004). Les auteurs présentent une formalisation des techniques de fragmentation traditionnelles pour une collection XML (horizontale, verticale et mixte) et des règles qui assurent une fragmentation correcte. Des expériences sur les trois types de fragmentation ont été menées sur un système nommé PartiX implémenté à l'occasion (Andrade et al., 2005).

Fragmentation XML dans le contexte distribué Bremer et Gertz proposent une approche de distribution de données XML par fragmentation (Gertz & Bremer, 2003). Les auteurs proposent une fragmentation verticale et une autre horizontale. La spécification des fragments XML est basée sur un langage d'expression de chemins dérivé de XPath, nommé XF. Un fragment vertical est le résultat de l'évaluation d'une expression XF sur un schéma global des données XML, nommé RG. Afin de garantir la cohérence et la disjonction des fragments, les auteurs associent à chaque expression de sélection une expression d'exclusion. Ces expressions définissent les fragments à exclure ou à ignorer. Un fragment horizontal est obtenu par une expression XF qui intègre des conditions sur les éléments, attributs et valeurs et utilise la notion de (*branching path expression*) (Gertz & Bremer, 2003). La figure 2.5 présente un exemple de fragmentation verticale selon Bremer et Gertz. Le fragment $f1$ est obtenu par l'expression $XF := //A//C$, et le fragment $f2$ est exclu selon l'expression d'exclusion $//D/*/E$ (Bremer & Gertz, 2003).

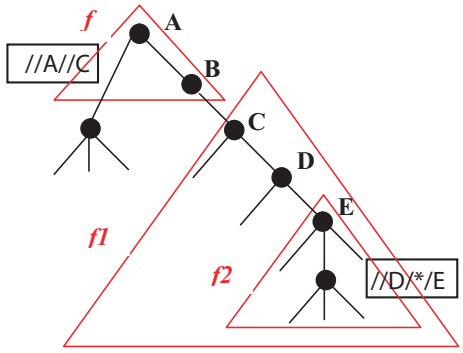

FIG. 2.5 – Exemple de fragments verticaux selon Bremer et Gertz

Bose et Fegaras utilisent des fragments XML pour le transfert de données dans une architecture pair-à-pair (Bose & Fegaras, 2005). Chaque fragment est identifié par un identificateur *ID* et est obtenu par une expression de chemin. Les fragments sont aussi reliés entre eux. Les relations entre fragments sont définies par les concepts *holes* et *fillers*. *Holes* indique qu'un élément d'un fragment est la racine d'un sous-arbre, un *fillers*. Les auteurs proposent un schéma de fragmentation, nommé *Tag Structure*, qui définit la structure des données et fournit des informations sur la fragmentation. Ces informations sont utilisées pour le traitement des requêtes sur les fragments et permettent aussi d'identifier les fragments à charger en mémoire.

Bonifati *et al.* modélisent des fragments XML verticaux pour une architecture XML pair-à-pair (Bonifati et al., 2004). Un fragment XML est obtenu et identifié par une expression de chemin (XP, *root-to-node path expression*). Il est ensuite stocké dans un pair. De plus, deux types d'expressions de chemins sont associées

à chaque fragment : des expressions de chemins *super fragments* et *child fragments* (Figure 2.6). Cela permet d'identifier les différents sites ou fragments nécessaires pour le traitement d'une requête ou l'expression de chemins sur l'architecture.

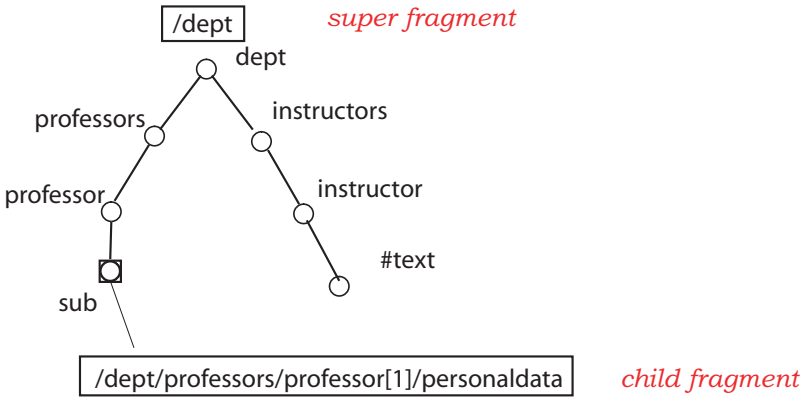

FIG. 2.6 – Exemple d'expressions de chemins selon Bonifati *et al.*

Travaux	**Définition des fragments**	**Règles de fragmentation**
Ma *et al.* (2003)	Expressions de chemin	+
Bonifati *et al.* (2007)		
Andrade *et al.* (2006)	Opérateur TLC	
Bremer et Gertz (2003)	Expressions XF	+
Bonifati *et al.* (2004)	Expressions XP	
Bose et Fegaras (2005)	Expressions de chemin	

TAB. 2.4 – Fragmentation des bases de données XML

Synthèse Les travaux de fragmentation dans le contexte des bases de données XML sont résumés dans le tableau 2.4. Ces travaux diffèrent dans la manière de définir formellement les fragments : algèbre TLC, expressions de chemin XF ou langage XML-QL. De plus, certains travaux définissent également des règles pour une fragmentation correcte : la complétude, la disjonction et la reconstruction (Gertz & Bremer, 2003; Ma et al., 2003). Ces règles garantissent une décomposition syntaxiquement correcte des requêtes distribuées et permettent d'assurer un résultat cohérent lors de la reconstruction.

2.2.3 Fragmentation basée sur la fouille de données

Les techniques de fouille se sont montrées efficaces pour la sélection de structures de données aidant à améliorer les performances d'un système de gestion de bases de

données, comme les index et les vues matérialisées (Aouiche et al., 2006; Azefack et al., 2007). Par ailleurs, plusieurs travaux exploitent des techniques de fouille de données pour la fragmentation.

Travaux de Darabant et Campan (2004) Les auteurs proposent une méthode de fragmentation horizontale d'une base de données orientée objets distribuée (Darabant & Campan, 2004). Ils se basent sur une classification par la technique des k-means (Pham et al., 2004). Cette technique classe les instances d'objets dans des fragments en tenant compte des relations entre les classes (agrégation, associations et liens entre méthodes) et regroupe les objets similaires en se basant sur des conditions extraites des requêtes utilisateurs. Pour cela, les auteurs proposent des fonctions de similarité entre objets calculées selon différentes métriques et des méthodes qui permettent de choisir une distribution de classes initial selon la sémantique des requêtes.

Dans d'autres travaux, Darabant (2005) propose d'utiliser cette technique pour fragmenter une base de données orienté objets avec des attributs et des méthodes complexes. L'auteur définit un attribut complexe par un attribut de type ensemble ou intervalle, et une méthode complexe comme une méthode qui fait appel à une autre méthode d'une autre classe. L'auteur propose dans ces travaux une fragmentation horizontale dérivée.

Dans une base de données orientée objet, cette fragmentation s'applique en deux étapes : (1) une fragmentation primaire qui groupe les instances de classes selon des conditions définies sur les attributs des classes ; et (2) une fragmentation dérivée qui regroupe les instances d'une classe selon les fragments des classes mères. L'algorithme de fragmentation proposé par l'auteur prend en compte les relations entre les classes (agrégation, associations et liens entre méthodes complexes) et vise à unifier les étapes de fragmentation horizontale dérivée (primaire et dérivée) en une seule.

Travaux de Gorla et Betty (2005) Les auteurs exploitent les règles d'association pour la fragmentation verticale d'une base de données relationnelle (Gorla & Betty, 2008). Ils affirment que les performances du système peuvent être optimisées en réduisant le nombre d'accès aux données utiles pour une requête. Pour cela, ils adaptent l'algorithme *Apriori* (Agrawal & Srikant, 1994). Ils valident leur proposition avec deux bases de données réelles en faisant varier les seuils de support et confiance.

Travaux de Fiolet et Toursel (2005) Les auteurs proposent un algorithme de classification parallèle et progressive pour fragmenter une base de données et distri-

buer les fragments sur une grille (Fiolet & Toursel, 2005b). Cet algorithme est inspiré de l'algorithme de classification séquentiel *CLIQUE* qui consiste à classer les données par projection. Les fragments obtenus par classification permettent d'améliorer le traitement des requêtes parallèles et de minimiser le coût d'échange des données.

2.2.4 Synthèse générale

L'ensemble des travaux qui traitent de la fragmentation est résumé dans le tableau 2.5. Nous les comparons selon le type de fragmentation adopté : verticale, horizontale et mixte. Dans ce tableau, un « + » indique que l'approche utilise ou traite un type de fragmentation. P désigne la fragmentation horizontale primaire et D la fragmentation horizontale dérivée.

Les travaux développés dans le contexte des entrepôts de données relationnels sont inspirés de ceux proposés dans le contexte des bases de données transactionnelles. Certains visent à minimiser le coût de stockage et adoptent la fragmentation verticale pour matérialiser des index ou pour partitionner des vues définies sur l'entrepôt. D'autres travaux, plus récents, favorisent la fragmentation horizontale dérivée. Dans un contexte centralisé, cette fragmentation réduit le coût de traitement des requêtes de jointure en éliminant l'accès aux données inutiles. Cette fragmentation est également la plus adaptée pour la distribution. En effet, elle partitionne un entrepôt en sous-schémas en se basant sur des besoins de sites distants exprimés par des requêtes.

Les travaux proposés dans le contexte des bases de données XML sont inspirés de ceux développés dans les contextes relationnel et orienté objets. Ceux qui traitent les flux de données utilisent une fragmentation verticale et ne proposent pas de fragmentation horizontale. Or, cette dernière peut être plus performante dans le processus de reconstruction des résultats des sous-requêtes d'une requête distribuée. Les travaux qui proposent une distribution de données XML adoptent une fragmentation horizontale uniquement. En effet, des fragments horizontaux répondent aux besoins exprimés par des requêtes de sélection sur des sites distribués. Aucune approche de fragmentation horizontale dérivée n'est proposée. Cela est dû au fait que ces approches ne s'intéressent pas à la fragmentation de plusieurs documents XML reliés entre eux.

Finalement, les approches de fragmentation basées sur la fouille de données sont peu nombreuses, mais montrent toutefois que les techniques de fouille peuvent être utilisées pour la fragmentation verticale et horizontale, à l'aide de règles d'association (Gorla & Betty, 2008) et d'algorithmes de classification (Darabant & Campan, 2004), respectivement.

Technique	Travaux	Fragmentation		
		Horizontale		Verticale
		P	D	
Entrepôts de données relationnels	Wu and Buchmaan (1997)		+	+
	Golfarelli *et al.* (1999)			+
	Datta *et al.* (1999)			+
	Noaman *et al.* (1999)			+
	Munneke *et al.* (1999)		+	
	Bellatreche and Boukhalfa (2005)		+	
Données XML	Ma *et al.* (2003)	+		+
	Bonifati *et al.* (2007)			+
	Andrade *et al.* (2006)	+		+
	Bremer et Gertz (2003)	+		+
	Bose et Fegaras (2005)			+
Fouille de données	Gorla et Betty (2005)			+
	Darabant et Campan (2004)		+	
	Fiolet et Toursel (2005)			+

TAB. 2.5 – Comparaison des travaux de fragmentation des données

2.3 Distribution de données sur grille

La distribution consiste à répartir un ensemble de fragments sur une architecture distribuée (classique, pair-à-pair ou grille). Elle favorise la répartition d'un grand volume de données sur plusieurs sites ainsi que le traitement parallèle des requêtes (Bellatreche, 2000).

Dans ce contexte, les grilles forment une classe de systèmes à part, car elles réunissent des ressources hétérogènes et distantes à l'aide de services d'abstraction, tout en étant souvent entièrement dépourvues de gestion centrale (Wehrle et al., 2005). Elles fournissent aussi un accès transparent à un ensemble de ressources hétérogènes et autonomes (Gounaris et al., 2007). Nous présentons dans cette section les travaux qui traitent de la distribution des bases de données et des entrepôts relationnels sur grille, ainsi que ceux qui proposent de construire une grille pour des données XML.

2.3.1 Bases de données sur grille

Concevoir et construire une base de données répartie sur grille induit plusieurs challenges (Watson, 2005). En effet, les sites de la grille peuvent exploiter différents produits de bases de données (Oracle, DB2...), voire différents paradigmes (relationnel, orienté objet, XML). Les travaux proposés dans ce contexte visent à fournir

un ensemble de logiciels génériques (intergiciels) qui supportent la fédération d'une base de données sur grille. Ces travaux proposent également des solutions d'intégration des schémas de bases de données hétérogènes. Cette intégration se base sur des métadonnées qui peuvent être représentées par des ontologies (Matsuda, 2005). Des mécanismes d'interrogation et de traitement parallèle de requêtes et de flux de données sont aussi proposés.

Travaux de Watson (2005) L'auteur propose une architecture basée sur les services (Watson, 2005). Ces services sont déployés sur les sites de la grille et facilitent l'échange et l'interrogation des données. Ils permettent l'extraction et le formatage (graphique par exemple) des données d'un site. Ils matérialisent également le résultat d'une requête sur un site pour une future interrogation et combinent les métadonnées de chaque site avec d'autres services qui implémentent des opérateurs relationnels (jointure, union...). Le système proposé considère un SGBD comme un composant fondamental de la grille.

Travaux de Nieto-Santisteban *et al.* (2005) Les auteurs étudient les avantages de combiner les bases de données et les technologies associées aux grilles (Nieto-Santisteban et al., 2005). Pour cela, ils comparent MaxBCG, une grille classique basée sur un système de fichiers, avec une implémentation SQL équivalente qui regroupe des serveurs SQL sur une grille. Les auteurs évaluent plus précisément le traitement d'un algorithme spécifique au domaine de l'astronomie en exploitant les deux architectures : grille classique et bases de données sur grille. Cette algorithme exploite des données réparties sur des sites distants et nécessite un traitement parallèle. Les résultats de cette évaluation montrent qu'une grille exploitant des bases de données et le langage SQL permet de meilleures performances qu'une grille classique pour le traitement d'algorithmes complexes.

Travaux de Hideo (2005) L'auteur propose une approche pour l'intégration de bases de données distribuées dans une grille (Matsuda, 2005). Ces bases sont dédiées à des domaines différents. L'auteur se base sur une implémentation de grille existante, Toolkit Globus[5] et propose un système composé de trois modules :

1. un module de construction des métadonnées qui homogénéise les métadonnées des sites en utilisant les ontologies ;
2. un module de stockage des résultats des requêtes et des métadonnées ;

[5]http://www.globus.org/

3. un moteur de flot de données (*dataflow engine*) pour interroger les sites de la grille.

2.3.2 Entrepôts de données sur grille

Actuellement, plusieurs travaux qui traitent de la conception et de la construction des entrepôts de données distribués exploitent les technologies de grille (Costa & Furtado, 2007). En effet, la distribution d'un entrepôt de données sur une grille vise à satisfaire les besoins croissants en volume de stockage et en capacité de traitement.

Distribution d'entrepôts de données sur grille

Travaux de Wehrle *et al.* (2005) Les auteurs proposent modèle pour la distribution et l'interrogation d'un un entrepôt sur une architecture de grille (Wehrle et al., 2005) en exploitant la fragmentation horizontale dérivée. Pour cela, les auteurs se basent sur le concept de *chunk* (Deshpande et al., 1998). Un *chunk* représente une unité de données et facilite l'échange de données entre les sites de la grille. Cette unité est identifiée par une combinaison d'identificateurs de dimensions et stocke la valeur de la mesure correspondante. Les auteurs définissent un fragment horizontal par un ensemble de *chunks* appelé *block of chunks*.

Afin d'identifier et d'interroger un site de la grille, les auteurs exploitent un index basé sur la structure X-tree (Berchtold et al., 1996). X-tree est à l'origine un index spatial permettant d'indexer des points dans un espace dimensionnel.

Finalement, les auteurs exploitent un ensemble de services (Wehrle et al., 2007) :
- un service d'indexation LIS (*Local Indexing Service*) qui permet d'indexer les données de chaque site ;
- un service de pilotage et de communication CMS (*Communication and Monitoring Services*) qui aide à identifier les différents sites qui contiennent les données requises pour répondre à une requête ;
- un service de traitement de requête CRS (*Chunk Resolution Service*) qui permet d'identifier les fragments stockés dans un site.

Travaux de Costa et Furtado (2007) Les auteurs proposent le système Grid-DWPA (*Grid-Data Warehouses Parallel architecture*) (de Carvalho Costa & Furtado, 2006; Costa & Furtado, 2007). Grid-DWPA permet d'implanter un entrepôt de données parallèle dans un envirenement de grille. Pour cela, les auteurs définissent des paramètres de qualité de service (QoS, *Quality of Service*) sur lesquels ils se basent pour contrôler l'interrogation de l'entrepôt distribué (priorité et temps ré-

servé pour le traitement des requêtes). De plus, les auteurs définissent des stratégies qui visent à exécuter une sous-requête sur un site.

Le système Grid-DWPA déploie sur chaque site les composants suivants :

- un composant *Client* qui offre l'accès aux données stockées sur le site ;
- un composant *Exécuteur* qui permet d'interroger et d'échanger les données d'un site ;
- un composant *Chargeur de données* qui assure le chargement et la mise à jour des données sur un site.

Dans le système Grid-DWPA, la fragmentation et la répartition (*data placement*) des données sont définies selon la stratégie NPDW (*Node-Partitioning Data Warehouse*) (Furtado, 2004). Cette stratégie ne fragmente que les relations volumineuses (en se basant sur la charge du système) et réplique les petites relations sur chaque site de la grille. La replication des données est effectuée par une stratégie nommée PRG (*Partitioned Replica Groups*). Elle consiste à dupliquer sur chaque site les relations non volumineuses pour chaque groupe de nœuds.

Travaux de Gorawski *et al.* (2007) Les auteurs proposent le système GDWSA(t) (*Grid Data Warehouse Software Agent*) qui se base sur des agents (Gorawski et al., 2007) et exploite la technologie des grilles pour distribuer, charger et interroger les données d'un entrepôt. Il utilise également les agents pour la communication entre sites et l'extraction d'informations sur un site. Les auteurs proposent aussi un mécanisme d'indexation adapté aux grilles. Il permet de créer des index spécifiques aux caractéristiques de chaque nœud de la grille. Pour cela, les auteurs attribuent à chaque nœud un ensemble de facteurs (vitesse du processeur, mémoire disponible, temps moyen de transfert de données...) qui permettent de sélectionner une structure d'index adaptée.

Intégration d'entrepôts de données sur grille

Par ailleurs, d'autres travaux s'intéressent à l'intégration d'entrepôts de données géographiquement distribués et hétérogènes sur une grille. Iqbal *et al.* proposent une plateforme basée sur les services Web pour intégrer des entrepôts relationnels hétérogènes dans une architecture de grille (Iqbal et al., 2003). Chaque nœud de la grille comporte un service grille qui définit comment un client interagit avec les services disponibles dans la grille. Ces services permettent de créer, gérer et échanger des informations sur la grille. La plateforme proposée exploite la technologie JAX-RPC (Sun-microsystems, 2002).

Fiser *et al.* proposent une architecture nommée *Grid-enabled OLAP server* (Fiser

et al., 2004) qui permet d'analyser des données définies par un ensemble de *chunks* et stockées sur les sites d'une grille. Les auteurs présentent un moteur séquentiel, basé sur les services Web, qui permet des opérations OLAP. *Grid-enabled OLAP server* fait partie de l'architecture *GridMiner* conçue pour permettre la fouille de données sur une grille (Fiser et al., 2004).

Finalement, Fiolet et Toursel exploitent les grandes capacités de calcul d'une grille pour la fouille de données (Fiolet & Toursel, 2005a, 2005b). La fouille est aussi exploitée pour une fragmentation permettant de réduire la communication et l'échange de données entre les sites de la grille. Les auteurs proposent un algorithme de classification pour le traitement de requêtes parallèles.

2.3.3 Données XML sur grille

Les travaux qui traitent des données XML sur grille sont peu nombreux. Dans une architecture de grille, XML est plutôt considéré comme un format de représentation de données de sources hétérogènes et employé pour l'échange de données et la soumission de tâches (requêtes et procédures) sur la grille.

Niemi *et al.* proposent un médiateur qui supporte les paradigmes relationnel et XML et permet d'appliquer des techniques de sécurité sur une grille (Niemi et al., 2003). Les auteurs implémentent leur prototype en Java et affectent à chaque site de la grille un agent Java. Gounaris *et al.* exploitent le language XML pour la formulation de requêtes ciblant des ressources hétérogènes sur une grille (Gounaris et al., 2007). Finalement, Wang *et al.* proposent un algorithme de jointure pour intégrer et gérer des données XML sur un environnement de grille (Wang et al., 2005). L'index de jointure proposé permet la fusion de données et le traitement des requêtes sur la grille.

2.3.4 Synthèse

Les travaux qui proposent de construire des bases de données sur grille exploitent des services et développent des modules pour le stockage et l'interrogation des données. Ces services facilitent la communication et le transfert de données entre les nœuds de la grille. L'interrogation des données est basée sur les métadonnées de chaque site, qui peuvent être représentées par des ontologies ou extraites par des services spécifiques. Ces travaux proposent une intégration de bases de données distribuées et ne s'intéressent pas à la distribution des données elle-même.

Dans le contexte des entrepôts de données relationnels, les travaux proposés sont récents. Ceux qui proposent une répartition des données diffèrent dans la manière

de formaliser les fragments et de les allouer aux sites utilisés. Ces travaux exploitent également des structures d'index pour identifier et interroger les données de la grille. Cependant, ils ne proposent pas d'outil d'analyse pour l'entrepôt distribué. Les travaux qui proposent l'intégration d'entrepôts répartis se basent sur des services et sont conçus pour des domaines spécifiques. Finalement, les travaux dans le contexte XML sont rares. En effet, XML est plus utilisé pour le transfert et la formulation de requêtes sur une grille (Gounaris et al., 2007; Niemi et al., 2003) que par le stockage.

Chapitre 3

Modèle de référence d'entrepôt de données XML

Nous jugeons nécessaire de disposer d'un modèle de référence pour les entrepôts de données XML. Divers modèles d'entrepôts de données XML sont proposés dans littérature. Ils diffèrent dans la manière de représenter les faits et les dimensions et le nombre de documents XML utilisés. De plus, aucun standard n'est défini pour modélisation des entrepôts de données XML.

Nous proposons donc dans ce chapitre un modèle de référence pour les entrepôts de données XML. Ce modèle synthétise et unifie les travaux qui traitent de l'entreposage de données XML (section 2.1). Il étend également les propositions avancées dans la littérature (section 2.1.1). De plus, nous comparons dans ce chapitre les stratégies de stockage de données XML et identifions la plus adaptée pour les données des entrepôts XML (section 3.3). Finalement, nous présentons le langage d'interrogation XML que nous souhaitons utiliser pour formuler des requêtes décisionnelles dans la section 3.4.

3.1 Motivation

Dans la section 2.1, nous avons présenté les travaux qui traitent de l'entreposage des données XML. Nous avons mentionné que ces travaux peuvent être classés en deux familles. La première famille propose traite des entrepôts de documents XML et considère un entrepôt comme une collection de documents matérialisés par des vues XML. Les propositions de cette famille sont orientées besoins et sont plutôt employées lorsque les spécifications de l'entrepôt sont peu susceptibles d'évoluer. La deuxième famille respecte la représentation multidimensionnelle classique des entrepôts de données et modélise explicitement les dimensions. Les travaux de cette

famille utilisent des documents XML pour représenter les faits et les dimensions, et convergent plus ou moins explicitement vers un modèle d'entrepôt de données XML dérivé du modèle en étoile.

Les propositions de cette famille s'avèrent donc les plus adaptées dans notre contexte. Toutefois, elles diffèrent par le nombre de documents XML utilisés et la manière de représenter les faits et les dimensions. D'après ces travaux, un entrepôt de données XML peut avoir différentes représentations :

1. une collection de documents XML, où chaque document stocke un fait et les instances des dimensions correspondantes (X-Warehousing) ;
2. un document XML qui regroupe tous les faits et un autre qui stocke toutes les instances des dimensions (XCube) ;
3. une collection de documents XML où chaque fait et chaque instance de dimension est stocké(e) dans un document XML séparé (XML-OLAP) ;
4. un document XML qui regroupe tous les faits et un document XML séparé pour chaque dimension (organisation analogue à un schéma en étoile dans le contexte relationnel).

Une évaluation de performance de ces différentes représentations d'entrepôts de données XML a été menée par Boukraa *et al.* (Boukraa et al., 2006). Les auteurs ont généré des entrepôts de données mammographiques qui respectent ces quatre représentations. Les documents XML de ces entrepôts ont été alimentés à partir de la base DDSM (*Digital Database for Screening Mammography*)[1] et stockés grâce au SGBD natif XML eXist (Meier, 2002). Les auteurs ont comparé les temps de traitement d'une charge de requêtes XML composée de neuf requêtes. Les cinq premières requêtes sont simples. Ce sont des requêtes XPath qui effectuent des restrictions. Les autres requêtes sont plus complexes et exploitent des opérations de jointure entre différents documents de l'entrepôt. Elles sont formulées en XQuery. Les résultats de cette évaluation ont montré que stocker les faits dans un document XML et chaque dimension dans un document XML distinct permet d'obtenir les meilleures performances lors de l'interrogation de l'entrepôt.

De plus, comme chaque dimension et ses différents niveaux hiérarchiques sont stockés dans un seul document XML, les opérations de mise à jour peuvent être traitées facilement et efficacement, ce qui n'est pas le cas quand les dimensions et les faits sont représentés dans un seul document XML (XCube) ou regroupés dans des documents XML relatifs à chaque fait (X-Warehousing).

[1] http://marathon.csee.usf.edu/Mammography/Database.html

Nous nous basons donc sur cette représentation pour modéliser notre entrepôt de données XML. Notre modèle représente les faits par un seul document XML et chaque dimension par un document XML (section 3.2).

3.2 Modèle unifié d'entrepôt de données XML

Nous proposons dans cette section un modèle unifié pour les entrepôts de données XML. Ce modèle synthétise et enrichit les modèles d'entrepôts de données XML présentés dans la section 3.1. Notre modèle d'entrepôt de données XML est présenté dans la figure 3.1. Il se base sur un schéma conceptuel en constellation. Notre modèle représente les faits par un seul document XML et chaque dimension par un document XML. Plus précisément, notre entrepôt de données XML est composé des documents XML suivants :

- *dw-model.xml* stocke les métadonnées de l'entrepôt, c'est-à-dire son schéma. Il décrit la structure du ou des documents faits (mesures et références aux dimensions) et dimensions (niveaux hiérarchique de chaque dimension et attributs qui les caractérisent).
- Un ensemble de documents XML $facts_f.xml$ permet de stocker l'extension des faits de la constellation, c'est-à-dire les identificateurs des dimensions et les valeurs des mesures.
- Un ensemble de documents XML $dimension_d.xml$ représente les instances de chaque dimension d.

Chacun de ces documents XML peut être représenté par un graphe XML (définition 8). Nous présentons et définissons ci-après la structure (éléments, attributs et hiérarchie) des différents documents XML qui constituent l'entrepôt de données par des graphes XML.

Définition 8 *Soient E l'ensemble des noms d'éléments distincts, A l'ensemble des noms d'attributs distincts d'un document XML et V l'ensemble des valeurs de ces éléments et de ces attributs. Un graphe de données XML peut se représenter par $\Gamma :=< t, l, \psi >$, où t est un arbre ordonné, l est la fonction qui étiquette un nœud de t avec des symboles appartenant à $E \cup A$ et ψ est la fonction de correspondance entre les nœuds de t et leurs valeurs dans V. Le nœud racine de l'arbre t est dénoté $root_t$.*

Les relations entre les niveaux hiérarchiques d'une dimension, ainsi que les relations entre faits et dimensions sont représentés par des clés de référence virtuelle équivalentes à des clés étrangères dans le contexte relationnel (définition 9).

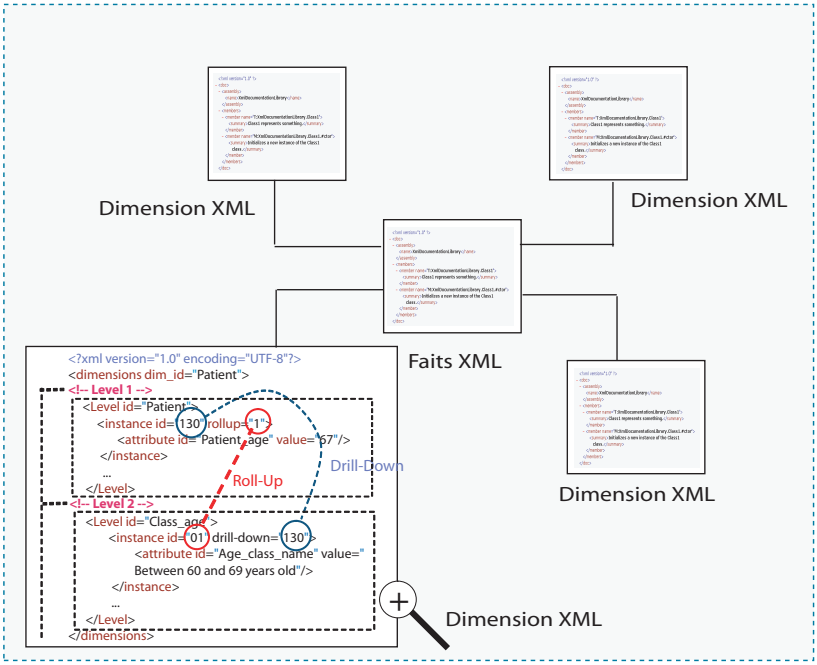

FIG. 3.1 – Architecture globale d'entrepôt de données XML

Définition 9 *Soit G un graphe XML et E l'ensemble de ses éléments. Une clé de référence virtuelle définit le lien entre deux nœuds $e \in E$ et $e' \in E$. Soient a et a' des attributs fils des éléments e et e', respectivement. e fait reference à e' si et seulement si $\psi(a) = \psi(a')$ and $l(a) \neq l(a')$.*

3.2.1 Document *dw-model.xml*

Le document *dw-model.xml* définit la structure multidimensionnelle de l'entrepôt de données. Le schéma *dw-model.xsd* qui valide ce document est disponible en annexe 1. La structure du graphe de ce document, $G_{dw-model}$, est présentée dans la figure 3.2. Dans ce graphe, les étiquettes des nœuds représentent les noms des éléments, celles contenant le caractère @ représentent les attributs.

Le nœud racine, *DW-model*, est composé d'un ou plusieurs nœuds *dimension* et *FactDoc*, qui décrivent respectivement les dimensions et les faits de l'entrepôt de données XML.

Un nœud *dimension* définit une dimension d ainsi que ses niveaux hiérarchiques. Il fournit aussi, par l'attribut *@path*, le chemin du document $dimension_d.xml$ correspondant. Un nœud *dimension* est constitué d'un ensemble de nœuds *Level* qui décrivent ses niveaux hiérarchiques. Un nœud *Level* stocke les noms (*@name*) et les types (*@type*) des attributs d'un niveau donné.

Un nœud *FactDoc* représente un ensemble f de faits. Il fournit leur nom (*@id*) et

le chemin du document XML qui les stocke (*@path*). Il décrit également les mesures et leurs types, par les nœuds *measure* et les dimensions qui caractérisent f par des nœuds *dimension*.

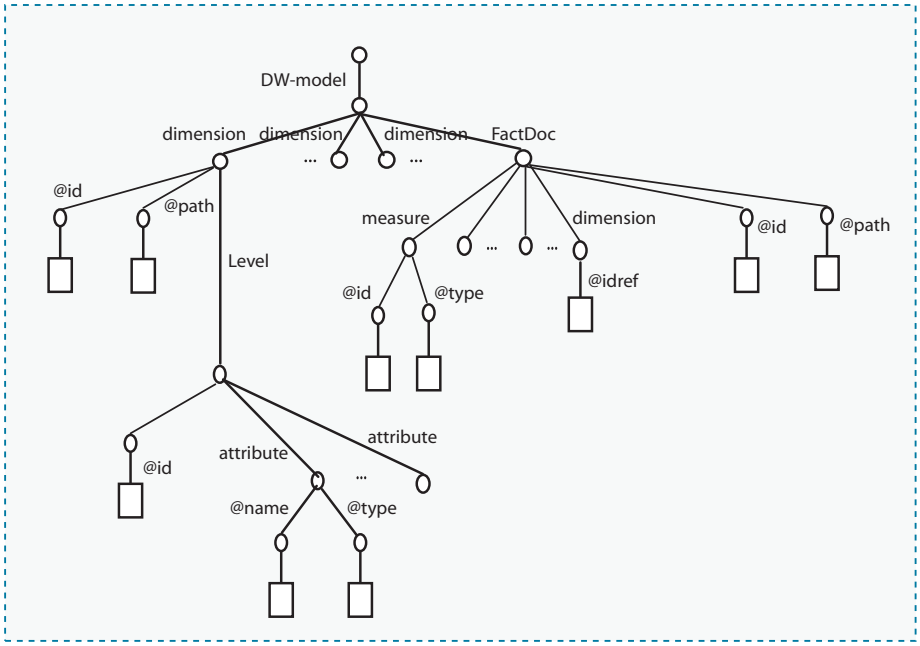

FIG. 3.2 – Structure du graphe $G_{dw-model}$

3.2.2 Documents $facts_f.xml$

Un document $facts_f.xml$ décrit un ensemble f de faits. La figure 3.4 (a) représente le graphe XML d'un document $facts_f.xml$, dénoté G_{facts_f}. Un document $facts_f.xml$ est valide selon le schéma XML *facts.xsd* disponible en annexe 2.

Le nœud racine de ce graphe, *FactDoc*, est composé de nœuds *fact*. Un nœud *fact* est constitué d'un ou plusieurs nœuds *measure* et d'un ensemble de nœuds *dimension*. Un nœud $fact$ possède également un attribut *@id* qui spécifie une instance de f (le nom du fait observé). Un nœud *measure* décrit une mesure par son nom (*@mes-id*) et sa valeur (*@value*). Un nœud *dimension* stocke le nom et l'identificateur d'une instance de dimension dans les attributs *@dim-id* et *@value-id*, respectivement.

Un exemple de document XML faits est présenté dans figure 3.3. Ce document stocke des faits ventes caractérisés par les mesures quantité des produits achetés (*Quantity*) et montant des produit achetés (*Amount*). Ces faits sont décrits par les dimensions livres (*Books*), consommateurs (*Customers*), géographie (*Geo*) et temps (*Date*).

Finalement, les faits du document $fact_f.xml$ peuvent présenter des structures irrégulières qui sont difficiles, voire impossibles à décrire dans le contexte relationnel.

Ces structures peuvent être définies comme suit.

- *Dimensions manquantes.* Les faits sont généralement décrits par toutes les dimensions. Cependant, un ou plusieurs faits peuvent présenter des dimensions manquantes. Par exemple, le premier fait du document $facts_{sales}.xml$ de la figure 3.3 est décrit par les dimensions *Book, Customers, Geo et Date*, tandis que le dernier est décrit uniquement par la dimension *Geo*.
- *Ordre des dimensions variable.* L'ordre des dimensions qui décrivent les faits peut également varier d'un fait à un autre. Par exemple, dans la figure 3.3, le premier fait est décrit par les dimensions : *Book-Customers-Geo-Date*. Tandis que le second est décrit par les dimensions : *Book-Date-Geo-Customers*. Cet ordre est important dans un document XML.
- *Niveaux de granularité variable.* Les fait peuvent être décrits par des dimensions à différents niveaux de granularité. Par exemple, dans le document $facts_{sales}.xml$ de la figure 3.3, le premier fait est décrit par la dimension *Book*, tandis que le troisième fait est décrit par la dimension *Book/Categories*.

3.2.3 Documents $dimension_d.xml$

Cet ensemble de documents stocke les dimensions de l'entrepôt de données XML. Une dimension d est représentée par un document $dimension_d.xml$. La structure du graphe de ce document, noté $G_{dimension_d}$, est présentée dans la figure 3.4 (b). Le schéma *dimension.xsd* qui le valide est disponible en annexe 3.

Le nœud racine de ce graphe, *dimension*, est constitué d'un ensemble de nœuds *Level* qui caractérisent les différents niveaux hiérarchiques de la dimension d. Un nœud *Level* est composé de nœuds *instance* qui définissent les instances de ce niveau. De plus, un nœud *dimension* possède deux attributs *@Roll-up* et *@Drill-Down* qui définissent les relations entre les différents niveaux hiérarchiques de la dimension d (Figure 3.4 (3)). *@Drill-Down* relie par exemple une instance d'un niveau hiérarchique aux instances correspondantes d'un niveau hiérarchique de granularité inférieure. Les attributs *@Roll-up* et *@Drill-Down* sont multivalués, ce qui permet de définir la structure irrégulière des faits et des dimensions.

Un exemple de document $dimension_d.xml$ est présenté dans la figure 3.5. Ce document représente la dimension livres (*Books*) organisée en deux niveau hiérarchiques : Livres → Catégories (*Books* → *Book-categories*).

Un document $dimension_d.xml$ permet également de représenter des structure de hiérarchies complexes (*ragged hierarchies*). Dans le contexte XML, ce concept a été introduit par Beyer *et al.* (2004) et fait référence à des hiérarchies de dimension irrégulières. Une telle hiérarchie peut présenter une ou plusieurs des caractéristiques

```
<?xml version='1.0' encoding='UTF-8'?>
<FactDoc id="sales">
     <fact>
      <dimension dim-id="Books" value-id="KauTra93"/>
      <dimension dim-id="Customers" value-id="C3"/>
      <dimension dim-id="Geo" value-id="1"/>
      <dimension dim-id="Date" value-id="01/01/98"/>
      <measure mes-id="Quantity" value="150"/>
      <measure mes-id="TotalAmount" value="8850.00"/>
     </fact>
     <fact>
      <dimension dim-id="Books" value-id="KauTra97"/>
      <dimension dim-id="Date" value-id="01/01/98"/>
      <dimension dim-id="Geo" value-id="1"/>
      <dimension dim-id="Customers" value-id="C3"/>
      <measure mes-id="Quantity" value="280"/>
      <measure mes-id="TotalAmount" value="4550.00"/>
     </fact>
     <fact>
      <dimension dim-id="Books-Categories" value-id="Gat-Java"/>
      <dimension dim-id="Customers" value-id="C56"/>
      <dimension dim-id="Geo" value-id="2"/>
      <dimension dim-id="Date" value-id="03/01/98"/>
      <measure mes-id="Quantity" value="151"/>
      <measure mes-id="TotalAmount" value="780.00"/>
     </fact>
     <fact>
      <dimension dim-id="Customers" value-id="C5"/>
      <dimension dim-id="Books" value-id="KauTra94"/>
      <dimension dim-id="Date" value-id="06/01/98"/>
      <dimension dim-id="Geo" value-id="3"/>
      <measure mes-id="Quantity" value="100"/>
      <measure mes-id="TotalAmount" value="5900.00"/>
     </fact>
     ...
     <fact>
      <dimension dim-id="Geo" value-id="7"/>
      <measure mes-id="TotalAmount" value="17900.00"/>
     </fact>
</FactDoc>
```

FIG. 3.3 – Exemple de document faits : $fact_{sales}.xml$

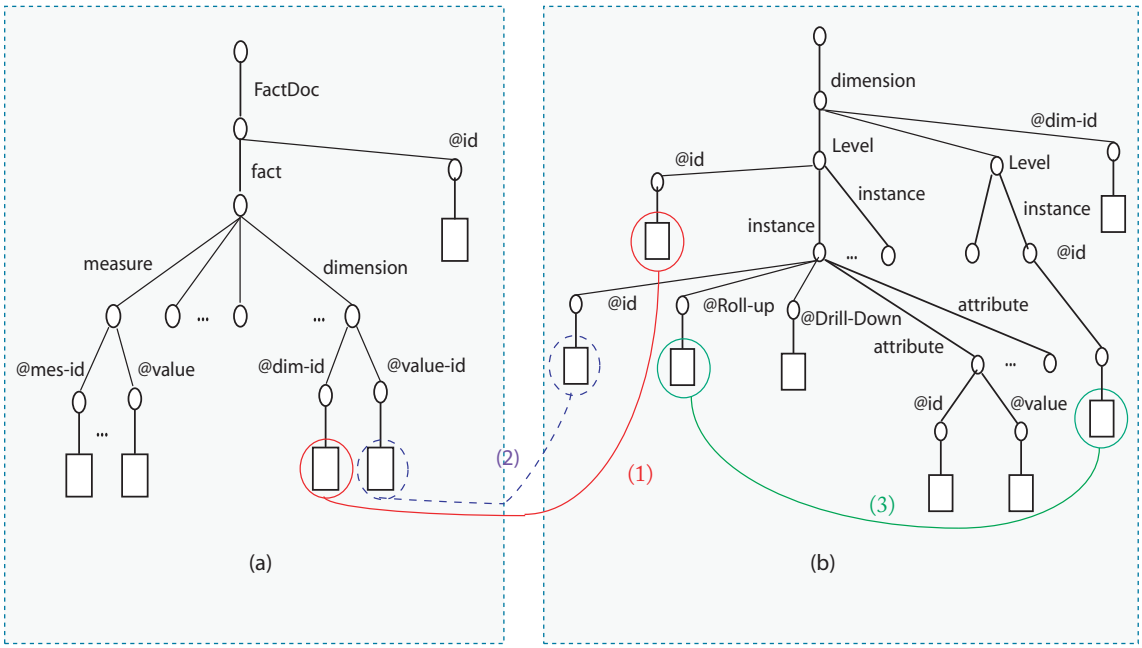

FIG. 3.4 – Structures des graphes (a) G_{facts_f} et (b) $G_{dimension_d}$

suivantes.

- *Hiérarchie non-stricte*. Dans ce type de hiérarchie, les instances d'un niveau peuvent être regroupées dans une ou plusieurs instances d'un niveau plus général. Cela se traduit par une association plusieurs-à-plusieurs entre deux niveaux hiérarchiques d'une dimension (Torlone, 2003). Par exemple, un livre peut appartenir à une ou plusieurs catégories et une catégorie peut regrouper un ou plusieurs livres. Dans le document $dimension_{books}.xml$ de la figure 3.5, le livre *Transaction Processing* (première instance du niveau *Book*) appartient aux catégories *Concurrency* et *Distributed*.
- *Hiérarchie récursive*. Ce type de hiérarchie a été défini par Rizzi (2007). Dans une telle hiérarchie, les instances d'un niveau peuvent être regroupées dans une instance du même niveau. Par exemple, dans le document $dimension_{books}.xml$, les livres de la catégorie *Cat-DB* sont aussi des livres de la catégorie *Cat-Software*.
- ***Hiérarchie incomplète***. Dans ce type de hiérarchie, le niveau de granularité peut varier. Par exemple, des livres peuvent être regroupés par catégories ($Book \rightarrow Categories$), tandis que d'autres peuvent être regroupés par catégories et par domaine ($Book \rightarrow Categories \rightarrow Domain$).

L'intérêt d'XML dans ce cadre est de pouvoir prendre en compte tous ces types de hiérarchies à la fois.

```
<?xml version="1.0" encoding="UTF-8" ?>
<dimension dim-id="Books">
     <level id="Books">
          <instance id="KauTra93" roll-up="Cat-Con Cat-Dis">
               <attribute id="b_title" value="Transaction Processing"/>
               <attribute id="b_pub" value="Morgan Kaufmann"/>
               <attribute id="b_pub_year" value="1993" />
               <attribute id="b_price" value="59.00" />
          </instance>
          ...
          <instance id="ClePma97" roll-up="Cat-MIS Cat-PMa">
               <attribute id="b_title" value="Project Management"/>
               <attribute id="b_pub" value="Crisp Learning"/>
               <attribute id="b_pub_year" value="2002" />
               <attribute id="b_price" value="80.00" />
          </instance>
     </level>
     <level id="Book-categories">
          <instance id="Cat-Sof" drill-down="Cat-DB Cat-Jav Cat-Dis">
               <attribute id="name" value="Software" />
          </instance>
          <instance id="Cat-Ant" drill-down="KauRea95">
               <attribute id="name" value="Anthology" />
          </instance>
          <instance id="Cat-CSc" drill-down="Cat-ISy ClePma97">
               <attribute id="name" value="Computer_Sciences"/>
          </instance>
          ...
          <instance id="Cat-DB" roll-up="Cat-Sof"
          drill-down="Cat-Con KauRea95 ClePrg99">
               <attribute id="name" value="DB"/>
          </instance>
          <instance id="Cat-Dis" roll-up="Cat-Sof" drill-down="KauTra93">
               <attribute id="name" value="Distributed"/> </instance>
          </instance>
          <instance id="Cat-ISy" roll-up="Cat-CSc" drill-down="Cat-MIS">
               <attribute id="name" value="Information_Systems"/>
          </instance>
          <instance id="Cat-PMa" roll-up="Cat-Man" drill-down="ClePma97">
               <attribute id="name" value="Project_Management"/>
          </instance>
          <instance id="Cat-Con" roll-up="Cat-DB" drill-down="KauTra93">
               <attribute id="name" value="Concurrency"/>
          </instance>
</dimension>
```

FIG. 3.5 – Exemple de document dimension : $dimension_{books}.xml$

3.3 Stockage des données de l'entrepôt

On peut distinguer deux types de stratégies pour stocker des documents XML dans une base de données. La première stratégie propose un stockage par un SGBD relationnel, les données XML étant enregistrées dans des tables. La deuxième stratégie propose un stockage natif des documents XML, c'est-à-dire que les documents XML préservent leur structure hiérarchique lors du chargement dans un SGBD natif XML. Dans cette section, nous présentons et comparons ces deux stratégies afin de choisir un système pour le stockage des documents XML qui constituent l'entrepôt de données.

3.3.1 Stockage relationnel

Cette stratégie est supportée par des systèmes relationnels étendus par des outils de traitement de données XML. Elle définit un *mapping* qui permet la transformation des données XML en relations. Elle est guidée par la DTD/le schéma XML qui valide le document et nécessite la création de tables multiples pour le stockage de la structure hiérarchique du document. L'interrogation des données est assurée par des requêtes SQL qui utilisent plusieurs opérations de jointures.

Des systèmes tels qu'Oracle, DB2 et SQL-Server permettent de créer des attributs de type XML (Chen et al., 2006; Oracle-Corporation, 2006; Rys, 2003). Ces systèmes sont dits compatibles XML. Un champ XML permet le stockage d'un objet XML dans une table (Chen et al., 2006). Plusieurs documents XML de structures différentes peuvent ainsi être enregistrés dans une seule table. Dans Oracle, des fonctions du langage SQL permettent l'extraction des nœuds d'un document XML stocké dans une table via des sous-requêtes XPath (Oracle-Corporation, 2006). De même, SQL-Server propose une fonction nommée *query* qui permet la formulation de requêtes XQuery avec le langage SQL.

3.3.2 Stockage XML natif

Les SGBD natifs XML permettent de stocker des documents XML sans modifier leur structure. Ces systèmes proposent des schémas de stockage génériques, c'est-à-dire qu'ils sont indépendants des DTD/schémas XML. Ces systèmes considèrent le document XML comme unité centrale de la base (comme une relation dans une base de données relationnelle). Ce type de stockage offre également la possibilité de stocker plusieurs documents XML de structures irrégulières et différentes. Physiquement, les document XML sont représentés par des arbres ou des graphes de

données et organisés en collections. Ces systèmes incluent également des moteurs de traitement de requêtes XML. Une requête XML parcourt la structure d'arbre qui représente le document XML, extrait une partie de ce document et produit le résultat sous forme de document XML. Ces systèmes sont souvent dotés d'API (*Application Programming Interfaces*) qui offrent la possibilité d'accéder et d'interroger les données XML de la base depuis des applications extérieures. Les systèmes eXist (Meier, 2005), TIMBER (Jagadish et al., 2002) et X-Hive (X-Hive-Corporation, 2007) sont des exemples de SGBD natifs XML.

3.3.3 Critères de choix

Nous évaluons et comparons dans cette partie les deux stratégies de stockage de données XML selon les critères suivants.

Formulation des requêtes. Les SGBD relationnels compatibles XML permettent la formulation de requêtes simples, via les langages XPath ou XQuery, qui ciblent un seul document XML. Ils présentent des limitations pour la formulation de requêtes de jointure entre plusieurs documents XML (Mahboubi et al., 2008). En effet, les données XML sont stockées dans des champs XML d'une table. Une requête SQL-XQuery utilisant la fonction *query* doit par exemple être exécutée pour chaque enregistrement. Cela n'offre pas la possibilité de parcourir conjointement deux documents XML stockés dans différents enregistrements pour une opération de jointure. Au contraire, les SGBD natifs XML exploitent des langages d'interrogation XML permettant une interrogation complexe de divers documents XML de structures variées (Mahboubi et al., 2008). En effet, ces langages offrent la possibilité d'exprimer des requêtes de jointures sur plusieurs documents XML.

Temps de traitement des requêtes. Les SGBD natifs XML exploitent des schémas de stockage adaptés pour le parcours de structures en arbre. Cette stratégie de stockage sauvegarde physiquement ensemble des documents entiers ou utilise des pointeurs physiques (plutôt que logiques) entre les différentes parties des documents. Elle permet ainsi d'accélérer la vitesse d'accès aux données de la base (Meier, 2005). Les SGBD relationnels compatibles XML ne disposent pas de telles structures et utilisent un parcours séquentiel pour retrouver les champs XML d'une relation et y accéder. Nous avons évalué les performances des systèmes relationnels et XML natifs avec un index de jointure spécifique aux entrepôts de données XML (Mahboubi et al., 2006b, 2008; Mahboubi & Darmont, 2009b). Les résultats que nous avons obtenus montrent que l'utilisation d'un système XML natif permet de meilleures performances d'interrogation qu'un système relationnel quand le volume de données devient important.

Taille des données à stocker. Les SGBD relationnels compatibles XML récents permettent de stocker un nombre important de documents XML dans une table. Cependant, un champ XML ne peut supporter un document XML volumineux (une centaines de kilo-octets au maximum). Un système natif XML permet de stocker des documents de plus grandes tailles (une dizaine de méga-octets).

Complexité des mises à jour. L'initiative XML : DB[2] propose un formalisme de mise à jour de données XML. Ce formalisme peut être supporté par un SGBD natif XML. Il permet d'ajouter, de supprimer ou de modifier les éléments et attributs d'un document XML sans avoir à reconstruire entièrement la structure du graphe correspondant. Ces fonctionnalités ne sont pas incluses dans les système relationnels.

En conclusion, bien que les SGBD natifs XML actuels offrent des performances modestes, ils présentent des avantages, par rapport aux systèmes relationnels, qui nous font penser qu'ils sont mieux adaptés pour le stockage et l'interrogation de documents XML. Par conséquent, nous les adoptons pour stocker des données XML, tout en nous inscrivant dans une démarche nécessaire d'optimisation de performance.

3.4 Interrogation de l'entrepôt

La plupart des SGBD natifs XML supportent un ou plusieurs langages de requêtes. Les langages de requêtes les plus répandus sont XPath et XQuery. Le langage XPath a été initialement conçu pour être utilisé par les technologies XSLT[3] et XPointer[4] qui définissent un langage de transformation de documents XML et un langage d'adressage, respectivement. XPath a été adopté par la suite comme comme un petit langage d'interrogation. Une requête XPath est définie par une expression de chemin permettant d'extraire un ensemble nœud(s) d'un document XML (W3C, 2007).

XQuery est un langage fonctionnel conçu pour l'interrogation de n'importe quelle source XML (Boag et al., 2004). Il inclut le langage XPath. Une requête XQuery est une composition d'expressions qui retournent une séquence d'éléments XML ou une erreur. Une requête XQuery est de type FLOWR (*For Let Order by Where Return*) et permet de faire appel à la récursivité. Le langage XQuery est supporté par la plupart des SGBD natifs XML et est actuellement une recommendation du W3C (*World Wide Web Consortium*)[5].

Une requête XPath est une requête simple qui ne permet pas de formuler des

[2]http://xmldb-org.sourceforge.net/projects.html
[3]http://www.w3.org/TR/xslt
[4]http://www.w3.org/TR/2001/WD-xptr-20010108/
[5]http://www.w3.org/TR/xquery/

requêtes décisionnelles de type sélection, jointure et agrégation. En revanche, une requête XQuery peut être plus complexe. Elle permet de formuler des requêtes sur plusieurs documents XML incluant des opérations de jointure et d'agrégation. Nous supposons donc l'utilisation du langage XQuery pour formuler des requêtes décisionnelles.

La figure 4.2 donne un exemple de requête XQuery décisionnelle de type agrégation, jointure et sélection. Elle retourne la moyenne des quantités de ventes des clients demeurant à Lyon. Elle réalise une jointure entre les documents $dimension_{customer}.xml$ et $fact_{sales}.xml$, une sélection et une opération d'agrégation (*sum*) sur la mesure quantité.

for \$x **in** document($fact_{sales}.xml$)//FactDoc/Fact,
\$y **in** document($dimension_{customer}.xml$)
//dimensions[@dim-id='Customer']/Level/instance
where \$y/attribute[@id='c_city']/@value='Lyon'
and \$x/dimension[@dim-id='Customer']/@value-id=\$y/@id
return name='cust_name', **aggregation**(sum, quantity)

FIG. 3.6 – Exemple de requête décisionnelle XQuery

3.5 Bilan

Nous proposons dans ce chapitre un modèle de référence pour les entrepôts de données XML. Ce modèle s'inspire des propositions avancées dans la littérature et les unifie. Il utilise des documents XML dont la structure minimise le parcours des données XML lors d'une interrogation et facilite leur mise à jour. À la différence des travaux existants, le modèle que nous proposons permet aussi la modélisation des différents schémas d'entrepôt en étoile, en constellation et en flocon de neige.

La structure des documents XML de notre modèle permet également la représentation de faits et de dimensions irréguliers qui sont difficiles voire impossibles à représenter dans un contexte relationnel.

Nous avons également opté pour un stockage natif des données de l'entrepôt XML. Ce type de stockage permet en effet d'enregistrer des données XML sans modifier leur structure et offre ainsi la possibilité d'exprimer des requêtes XML d'analyse complexes. Ce type de stockage permet aussi de meilleures performances lors d'une interrogation des données, comparé à un stockage relationnel.

Nous utilisons finalement le langage XQuery pour la formulation de nos requêtes d'analyse. En effet, ce langage s'affirme comme un standard pour l'interrogation des données XML et est actuellement supporté par la plupart des SGBD natifs XML.

A la différence des autres langages XML, XQuery facilite la formulation de requêtes qui utilisent des opérations de jointures sur de multiples documents XML.

Chapitre 4

Fragmentation des entrepôts de données XML

Comme nous l'avons évoqué dans la section 1.2, notre travail vise à traiter conjointement les problématiques du volume des entrepôts XML et de la performance des requêtes décisionnelles. Pour cela, nous proposons de répartir un entrepôt de données sur une architecture distribuée. Généralement, il est en premier lieu nécessaire de fragmenter un ensemble de données avant de le répartir.

Dans ce chapitre, nous nous intéressons à la fragmentation des entrepôts de données XML. À notre connaissance, il n'existe aucune approche pour cela dans la littérature. De plus, les approches de fragmentation des bases de données XML (Section 2.2.2) ne sont pas adaptées pour les entrepôts de données XML (Section 4.1).

Nous proposons dans ce chapitre un ensemble de techniques de fragmentation pour les entrepôts XML : des adaptations des méthodes existantes dans le contexte des entrepôts relationnels et une méthode originale basée sur une technique de fouille de données (de classification, plus précisément).

4.1 Motivation

Dans la littérature, les approches qui proposent une fragmentation pour les bases de données XML se limitent à la fragmentation horizontale primaire (section 2.2.2). Cette fragmentation est appliquée sur un seul document XML et vise à minimiser le nombre d'entrées/sorties lors du traitement d'une requête XML exprimée par des expressions de chemin simples. Or, les entrepôts de données XML exploitent des requêtes qui comportent de multiples opérations de jointure qui ciblent plusieurs documents à la fois. Ces travaux ne sont donc pas adaptés à notre contexte. En ce qui concerne les entrepôts de données XML, aucune technique de fragmentation n'a

été avancée à notre connaissance.

Comme les travaux qui traitent de la fragmentation dans les entrepôts de données relationnels recommandent une fragmentation horizontale dérivée (section 2.2.1), nous avons proposé dans un premier temps d'adapter les méthodes de fragmentation horizontale dérivée les plus efficaces (fragmentation primaire basée sur la construction des prédicats et sur les affinités, section 2.2.1) au contexte XML (Mahboubi & Darmont, 2009a).

Cependant, ces techniques produisent un nombre important de fragments, qui est de plus inconnu *a priori* et impossible à prévoir. Cela peut détériorer les performances du système et rendre le processus de maintenance des fragments très coûteux. Or, il est important de contrôler ce nombre, notamment quand on souhaite répartir M fragments sur N sites avec $M > N$.

Bellatreche et Boukhalfa proposent d'utiliser des algorithmes génétiques pour contrôler le nombre de sous-schémas obtenus par fragmentation (Bellatreche & Boukhalfa, 2005), tandis que Ma et Schewe proposent une stratégie pour la sélection d'un schéma de fragmentation pour une base de données XML (Ma et al., 2003). D'autres travaux exploitent des techniques de fouille de données (algorithmes de classification) pour appliquer une fragmentation sur un ensemble de données (Darabant & Campan, 2004; Gorla & Betty, 2008). Ces démarches visent à appliquer une fragmentation intelligente pour à fins de distribution ou d'optimisation de performance des requêtes (Gorla & Betty, 2008).

Nous nous inspirons de ces travaux et proposons une démarche de fragmentation des entrepôts de données XML basée sur la technique de classification k-means (Mahboubi & Darmont, 2008), notre idée étant de contrôler par le paramètre k le nombre de fragments.

4.2 Approche de fragmentation des entrepôts de données XML

4.2.1 Définitions préalables

Dans un entrepôt de données XML, la fragmentation horizontale dérivée partitione tous d'abord les graphes $G_{dimension_d}$ des dimensions en se basant sur une charge de requête W, puis fragmente les graphes G_{fact_f} en se basant sur les fragments des graphes $G_{dimension_d}$. Une fragmentation horizontale dérivée pour un entrepôt de données XML modélisé selon un schéma en étoile ou ses dérivés, nécessite trois informations (Bellatreche, 2000).

1. **Noms des graphes des dimensions et des graphes des faits**. Les graphes $G_{dimension_d}$ stockent les instances des dimensions. Ces instances possèdent des attributs qui sont référencés par des attributs des graphes G_{facts_f}. Les graphes $G_{dimension_d}$ sont des graphes propriétaires. Les graphes G_{facts_f} stockent les identificateurs des instances des dimensions. Ils sont appelés graphes membres.
2. **Nombre de fragments horizontaux des dimensions**. Il est calculé à partir du nombre de fragments horizontaux de chaque graphe de dimension. Ce nombre peut être déterminé par la formule : $N = \prod_{i=1}^{|D|} |frag_i|$, où $|D|$ désigne le nombre de dimensions et $|frag_i|$ le nombre de fragments de la dimension i.
3. **Qualification de la jointure**. Les fragments des graphes G_{facts_f} sont dérivés à partir des fragments horizontaux des graphes $G_{dimension_d}$. Ce processus est appliqué par des opérations de semi-jointure définies par une qualification qui explicite la relation entre les graphes $G_{dimension_d}$ et G_{facts_f}.

4.2.2 Principe général

La fragmentation que nous adoptons se base sur un ensemble P de prédicats de sélection extraits d'une charge de requêtes W (figure 4.1). Nous détaillons ce processus dans la section 4.2.3. Les différentes méthodes de fragmentation que nous proposons produisent un schéma de fragmentation (section 4.2.5) et construisent les fragments de l'entrepôt de données XML.

4.2.3 Extraction des prédicats de sélection

L'ensemble P des prédicats de sélection utilisé pour fragmenter les graphes $G_{dimension_d}$ est identifié et extrait par une analyse syntaxique de la charge W. Les prédicats de sélection (définition 10) font partie de la clause `Where` des requêtes XQuery.

Définition 10 *Un prédicat de sélection est défini par l'expression* $p := R_a\theta[value \mid \emptyset_{XPath}(R) \mid Q]$*, où* R_a *et* Q *sont des expressions de chemins (définition 11) et* R *utilise l'attribut* a *,* $\theta \in \{=, <, >, \leq, \geq, \neq\}$*,* $value \in Dom_{a_k}$ *où* Dom_{a_k} *est le domaine de définition de l'attribut* a *et* $\emptyset_{XPath}$ *est une fonction XPath (W3C, 2007).*

Définition 11 *Une expression de chemin* R *est une séquence :* $root_t/e_1/.../\{e_n/@a_k\}$*, où* $\{e_1, ..., e_n\} \in E$ *et* $@a_k \in A$*. L'expression* R *peut contenir le symbole* $*$ *qui indique un élément quelconque de* E *et le symbole* $//$*, qui indique une séquence d'éléments* $e_i/.../e_j$ *telle que* $i < j$*. Le symbole [i] peut aussi être affecté à un élément* e_i*. Il indique alors la position de l'élément dans l'arbre.*

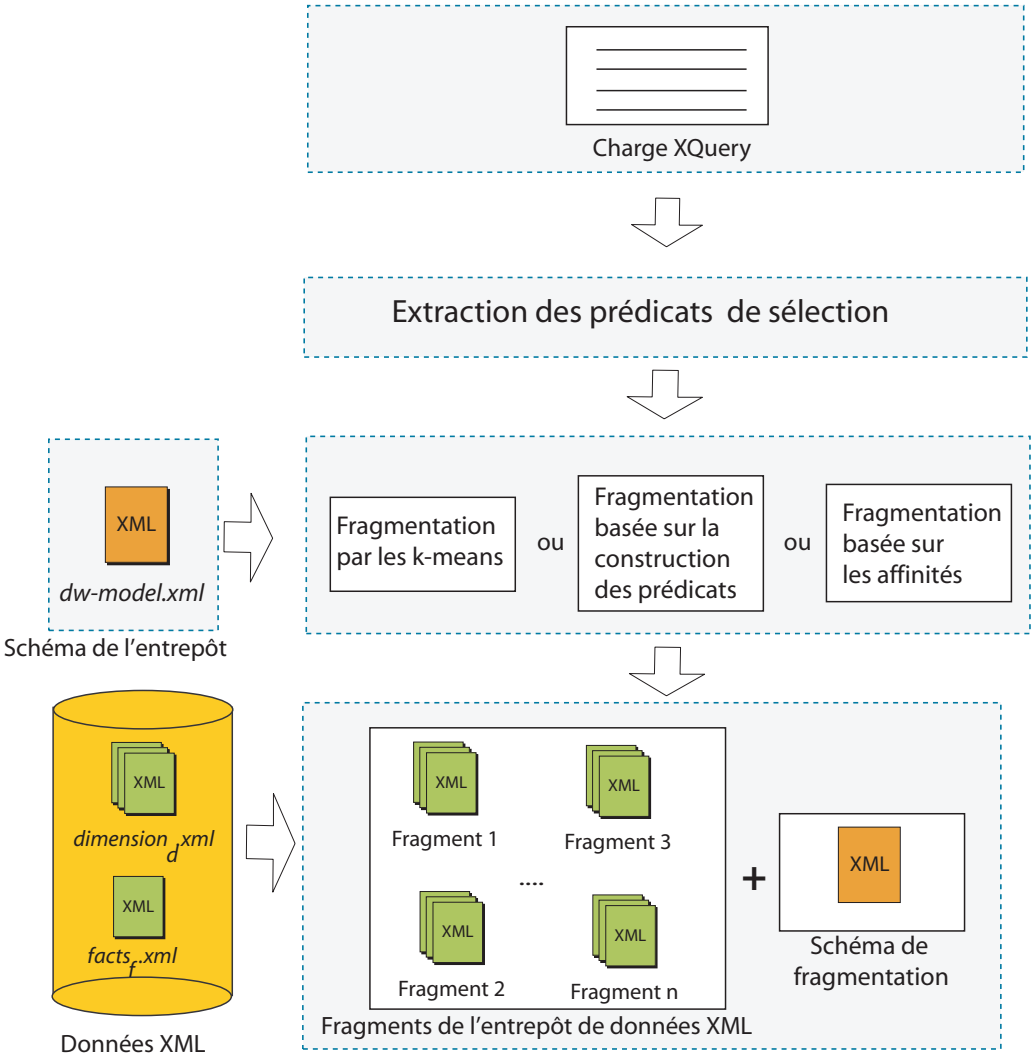

FIG. 4.1 – Principe de fragmentation d'un entrepôt de données XML

Exemple. $p_1 := \$y/attribute[@id =' c\_nation\_key']/@value >' 15'$
et $p_2 := \$y/attribute[@id =' p\_type']/\ @value =' PROMO\ \ BURNISHEDCOPPER'$
sont des prédicats de sélection obtenus depuis l'exemple de charge W_S de la figure 4.2.

4.2.4 Fragmentation des faits

Dans le cas de la fragmentation basée sur la construction des prédicats et de la fragmentation basée sur les affiniés, les graphes G_{facts_f} sont fragmentés en fonction des fragments des graphes $G_{dimension_d}$. La fragmentation des graphes G_{facts_f} est accomplie par des opérations de semi-jointure basées sur une clé de référence virtuelle (section 3.2). Cette clé assure la relation entre les graphes $G_{dimension_d}$ et G_{facts_f}. Elle définit explicitement la qualification de jointure présentée dans la figure 4.3. Cette qualification est constituée d'une conjonction de deux expressions de chemin. Ces expressions vérifient qu'un nœud *@value-id* (identificateur de la dimension *@dim-id* d'un fait) du graphe G_{facts_f} correspond à un nœud $@id$ (identificateur d'une instance de la même dimension) du graphe $G_{dimension_d}$.

4.2.5 Construction du schéma de fragmentation

Nous construisons finalement un document XML ($frag_{schema}.xml$) qui représente le schéma de fragmentation. Nous désignons par G_{schema} le graphe qui correspond à ce document. Sa structure est présentée dans la figure 4.4. Son nœud racine *Schema* est composé de nœuds *fragment* qui définissent les différents fragments primaires. Les nœuds *fragment* sont à leur tour composés de nœuds *dimension* qui représentent les fragments. Chaque nœud *dimension* est identifié par un nœud *@name* et composé de nœuds *predicate* qui définissent les prédicats de sélection correspondant à ce fragment.

q_1 **for** $x **in** //FactDoc/Fact,
$y **in** //dimensions[@dim-id='Customer']/Level/instance
where $y/attribute[@id='c_nation_key']/@value>'15'
and $x/dimension[@dim-id='Customer']/@value-id=$y/@id
return $x

q_2 **for** $x **in** //FactDoc/Fact,
$y **in** //dimensions[@dim-id='Customer']/Level/instance
$z **in** //dimensions[@dim-id='Part']/Level/instance
where $y/attribute[@id='c_nation_key']/@value='13'
and $y/attribute[@id='p_type']/@value='PROMO
BURNISHED COPPER'
and $x/dimension[@dim-id='Customer']/@value-id=$y/@id
and $x/dimension[@dim-id='Part']/@value-id=$z/@id
return $x

q_3 **for** $x **in** //FactDoc/Fact,
$y **in** //dimensions[@dim-id='Date']/Level/instance
where $y/attribute[@id='d_date_name']/@value='Monday'
and $x/dimension[@dim-id='Date']/@value-id=$y/@id
return $x

q_4 **for** $x **in** //FactDoc/Fact,
$y **in** //dimensions[@dim-id='Date']/Level/instance
where $y/attribute[@id='d_date_name']/@value='Saturday'
and $x/dimension[@dim-id='Date']/@value-id=$y/@id
return $x

...

q_{10} **for** $x **in** //FactDoc/Fact,
$y **in** //dimensions[@dim-id='Customer']/Level/instance
$z **in** //dimensions[@dim-id='Date']/Level/instance
where $y/attribute[@id='c_nation_key']/@value='7'
and $y/attribute[@id='d_date_name']/@value='Saturday'
and $x/dimension[@dim-id='Customer']/@value-id=$y/@id
and $x/dimension[@dim-id='Part']/@value-id=$z/@id
return $x

FIG. 4.2 – Exemple de charge : W_S

document($facts_f.xml$)/FactDoc/dimension[@dim-id=
document($dimension_d.xml$)/dimension/Level/@id]
and
document($facts_f.xml$)/FactDoc/dimension[@value-id
=document($dimension_d.xml$)/dimension/Level[@id
=@dim-id]/instance/@id]

FIG. 4.3 – Qualification de jointure

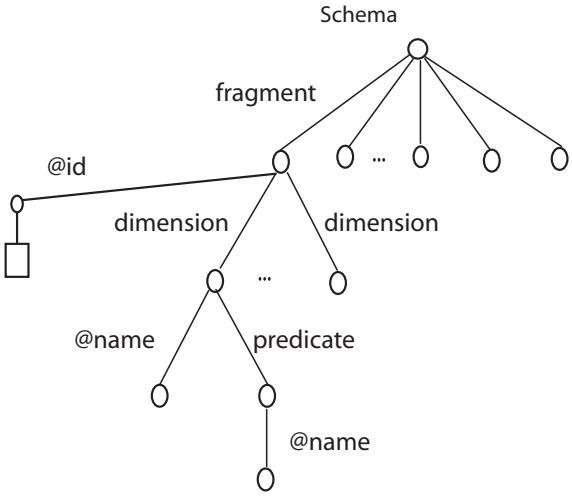

FIG. 4.4 – Structure du graphe G_{schema}

4.3 Fragmentation primaire basée sur la construction des prédicats (CP)

4.3.1 Principe

Cette méthode exploite l'ensemble P des prédicats de sélection ainsi que des métadonnées extraites de $G_{dw-model}$. Elle identifie tous d'abord les graphes candidats pour la fragmentation. Nous dénotons l'ensemble de ces graphes $G_{candidate_d}$. Un graphe $G_{candidate_d}$ est un graphe $G_{dimension_d}$ ciblé par une ou plusieurs requêtes de W. Pour chaque graphe candidat, CP identifie un sous-ensemble de prédicats $P_d \subset P$. Les prédicats de P_d sont ceux qui utilisent les attributs de la dimension d.

Pour chaque sous-ensemble P_d, CP génère ensuite un ensemble de prédicats complet et minimal, dénoté P'_d. Cet ensemble est obtenu par l'application de l'algorithme COM-MIN (Özsu & Valduriez, 1999). Cet algorithme génère des prédicats non redondants (minimalité) et permet de partitionner une dimension en au moins deux fragments séparément ciblés par une requête (complétude). L'ensemble P'_d permet finalement de créer des mintermes (définition 12) qui sont utilisés pour fragmenter les graphes $G_{candidate_d}$.

Définition 12 *Un minterme est la conjonction de prédicats de sélection de P et en utilisant les opérateurs $\wedge$ et $\neg$ (Ng & Seetharaman, 1997). Soit l'ensemble de prédicats $P = \{p, p'\}$, un minterme m peut être exprimé par l'expression $p \wedge p'$ ou $p \wedge \neg p'$.*

4.3.2 Étapes de la fragmentation

1. **Attribution des prédicats de sélection aux graphes des dimensions**. CP identifie dans un premier temps les prédicats de sélection propres à chaque graphe de dimension. Cette identification est effectuée en parcourant le graphe $G_{dw-model}$ et permet de construire pour chaque graphe de dimension un sous-ensemble de prédicats $P_d \subset P$. P_d regroupe les prédicats $p_i^d \in P$ utilisant des attributs membres de la dimension d. La méthode CP identifie ensuite les graphes de dimension candidats $G_{candidate_d}$ pour la fragmentation horizontale primaire. Un graphe $G_{candidate_d}$ est un graphe $G_{dimension_d}$ pour lequel l'ensemble P_d n'est pas vide. Nous disposons donc à l'issue de cette étape d'un ensemble de graphes $G_{candidate_d}$. Chaque graphe de cet ensemble sera fragmenté par des prédicats de P_d.

   Exemple. Les prédicats p_1 et p_2 extraits de la charge W_S (section 4.2.3) contiennent les attributs *c_nation_key* et *c_city*. Dans le graphe $G_{dw-model}$, ces attributs sont membres de la dimension *customer*. $P_{customer} = \{p_1, p_2\}$ est donc l'ensemble de prédicats qui correspond au graphe $G_{dimension_{customer}}$. Ce graphe est aussi identifié comme un graphe candidat pour la fragmentation.

2. **Complétude et minimalité des prédicats de sélection**.

   L'objectif de cette étape est de nous assurer qu'un graphe candidat est divisé en au moins deux fragments. Pour cela, nous appliquons l'algorithme COM-MIN (Özsu & Valduriez, 1999). Cet algorithme prend en entrée l'ensemble P_d et fournit en sortie l'ensemble complet et minimal $P_d^{'}$ de prédicats pertinents pour la fragmentation (algorithme 1). Il procède en deux étapes. La première étape (lignes 1 à 5) initialise l'ensemble $P_d^{'}$ avec un prédicat p et sa négation pour partitionner le graphe en deux fragments selon la règle 1 ci-dessous. La seconde étape est itérative et ajoute à $P_d^{'}$ tout nouveau prédicat p qui partitionne un fragment défini dans l'ensemble de résultats F en deux fragments selon la règle 1. les prédicats qui deviennent non pertinents par rapport à l'ensemble de résultats sont éliminés.

   Règle 1. Cette règle assure la complétude des prédicats. Elle vérifie qu'un prédicat p_i et sa négation partitionnent un graphe de dimension en au moins deux fragments.

En nous basant sur $P_d^{'}$, nous construisons ensuite l'ensemble des mintermes $M_d = \{m_i | m_i = \wedge_{q_j \in P} q_j^*\}$, où $q_j^* = q_j$ ou $q_j^* = \neg q_j$, $1 \leq j \leq n$, $1 \leq i \leq 2^n$ et n représente le nombre de prédicats de sélection. Le minterme $m_i \in M_d$ est la conjonction de tous les prédicats de $P_d^{'}$ sous forme normale et négative.

Algorithme 1 Algorithme de génération de $P_d^{'}$

Entrée : P_d
Sortie : $P_d^{'} \subset P_d$
Déclarations: F: ensemble de fragments, f: fragment défini par un minterm m

1: $P_d^{'} = \emptyset$
2: Chercher un prédicat $p \in P_d$ qui partitionne f tel que p respecte la règle 1
3: $P_d^{'} = \{p\}$
4: $P_d = P_d - \{p\}$
5: $F = \{f\}$
6: **répéter**
7: Chercher un prédicat $p^{'} \in P_d$ qui partitionne $f^{'} \in F$ en respectant la règle 1
8: $P_d^{'} = P_d^{'} \cup \{p^{'}\}$
9: $P_d = P_d - \{p^{'}\}$
10: $F = F \cup \{f^{'}\}$
11: **si** $\exists\ p_j \in P^{'}$ tel que p_j n'est pas pertinent **alors**
12: $P_d^{'} = P_d^{'} - \{p^{'}\}$
13: $F = F - \{f^{'}\}$
14: **fin si**
15: **jusqu'à** $P_d^{'}$ est complet

Exemple. Soit $P_{customer}^{'} = \{\$y/attribute[@id =' c\_nationkey']/@value = 13,$ $\$y/attribute[@id =' c\_nationkey']/@value > 15\}$ un ensemble de prédicats complet et minimal pour la dimension *customer*.
Le minterme m_1 est $\$y/attribute[@id =' c\_nationkey']/@value = 13$ *and* $\$y/attribute[@id =' c\_nation\ key']/@value <= 15$.

3. **Fragmentation des graphes candidats**. Dans cette étape, nous appliquons une fragmentation horizontale primaire sur $G_{candidate_d}$. Un fragment est obtenu en associant à chaque minterme $m_i \in M_d$ un ensemble de nœuds dans $G_{candidate_d}$. En effet, un minterme est la conjonction d'expressions de chemin. L'application de ces expressions sur le graphe $G_{candidate_d}$ produit donc un ensemble de nœuds qui forment le fragment.

Exemple. Le minterm m_1 est utilisé pour fragmenter $G_{candidate_{customer}}$. La structure du graphe qui représente le fragment obtenu est présenté dans la figure 4.5. Chaque nœud *instance* de ce graphe vérifie le minterm m_1.

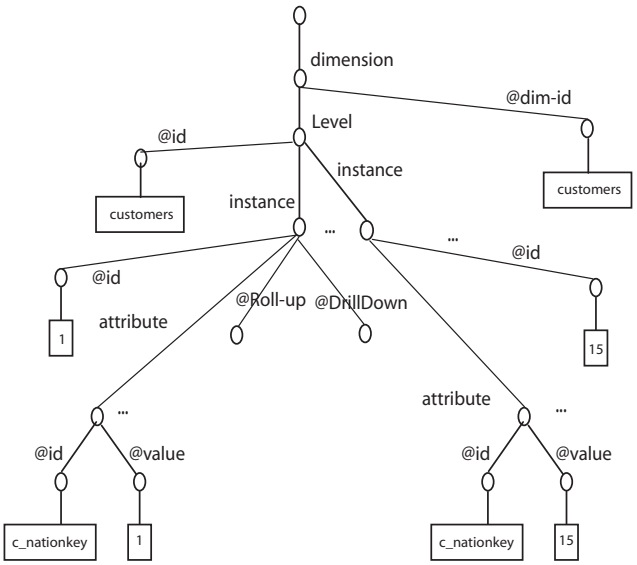

FIG. 4.5 – Exemple de fragment de dimension

4.4 Fragmentation primaire basée sur les affinités (BA)

4.4.1 Principe

En plus des prédicats de sélection, la méthode basée sur les affinités utilise les fréquences des requêtes de la charge pour construire les fragments des dimensions. Elle exploite des matrices spécifiques (matrice d'usage et matrice d'affinités) pour classer les prédicats de sélection de P. Pour cela, la méthode BA utilise un algorithme graphique. Une classe de prédicats regroupe des prédicats qui partagent des valeurs dans la matrice des affinités. Cette dernière est définie par un cycle de prédicats dans le graphe représentant la matrice des affinités et est utilisée pour construire un fragment d'un graphe $G_{dimension_d}$.

4.4.2 Étapes de la fragmentation

1. **Construction de la matrice d'usage des prédicats**. Nous nommons cette matrice PUM. Elle est construite à partir de l'ensemble P et définit l'usage de chaque prédicat $p_j \in P$ par chacune des requêtes $q_i \in W$. Les lignes de PUM représentent les requêtes et les colonnes les prédicats de sélection. Son terme général $PUM(i, j)$ est égal à un si q_i inclut le prédicat p_j, et à zéro sinon. De

plus, les fréquences des requêtes de W sont stockées dans un vecteur nommé $Freq$.

Exemple. Les tableaux 4.1 et 4.2 montrent la matrice PUM_S et le vecteur des fréquences $Freq_S$ correspondant à l'exemple de charge W_S (section 4.2.3), respectivement.

requête/prédicat	p_1	p_2	p_3	p_4	...	p_n
q_1	1	0	0	0		0
q_2	1	1	0	0		0
...						
q_m	1	1	0	0		1

n représente le nombre de prédicats de sélection dans P et m le nombre de requêtes dans W.

TAB. 4.1 – Exemple de matrice d'usage : PUM_S

q_1	q_2	...	q_m
10	20	...	5

TAB. 4.2 – Exemple de vecteur de fréquences : $Freq_S$

2. **Construction de la matrice d'affinité des prédicats**

Cette matrice est nommée Aff. Elle est construite à partir de la matrice PUM et du vecteur $Freq$. C'est une matrice carrée $n \times n$, où n représente le nombre de prédicats de sélection dans P. Les cellules de la matrice Aff peuvent avoir des valeurs numériques ou alphanumériques ($\Rightarrow$, $\Leftarrow$ et$*$).

- La valeur numérique d'une cellule $Aff(i, j)$ fournit la somme des fréquences des requêtes faisant référence aux prédicats p_i et p_j.
- La valeur $\Rightarrow$ indique que le prédicat p_i implique le prédicat p_j.
- La valeur $\Leftarrow$ informe que le prédicat p_j implique le prédicat p_i.
- La valeur $*$ indique que les prédicats p_i et p_j sont similaires. Deux prédicats p_i et p_j sont similaires si et seulement si (Navathe et al., 1995) :

  (a) ils exploitent le même attribut;

  (b) il existe deux requêtes q_a et q_b telles que q_a utilise les prédicats p_i et p_c et q_b utilise les prédicats p_j et p_c; p_c étant un prédicat de sélection qui utilise un autre attribut que celui utilisé par p_i et p_j.

Exemple. Le tableau 4.3 montre un exemple de matrice des affinités Aff_S construite à partir de la matrice PUM_S de la figure 4.1. Dans PUM_S, le prédicat p_2 est utilisé par les requêtes q_1 et q_2. La valeur de $Aff(2,2)$ est donc égale à 30. Elle correspond à la somme des fréquences des requêtes q_1 et q_2 (tableau 4.2).

	p_1	p_2	...	p_5	...	p_n
p_1	20	0		10		0
p_2	0	30				$\Leftarrow$
...						
p_5	10	0		25		0
...						
p_n	0	$\Rightarrow$		0		5

TAB. 4.3 – Exemple de matrice d'affinités : Aff_S

3. **Regroupement des prédicats**.

   Cette étape exploite un algorithme graphique proposé par Navathe *et al.* (1989) pour la fragmentation verticale et adapté à la fragmentation horizontale (Navathe et al., 1995). Cet algorithme prend en entrée la matrice Aff en la considérant comme un graphe complet, G_{Aff}. Il forme par la suite un *spanning tree*[1]. Un nœud de l'arbre représente un prédicat de sélection p_i dans Aff et un arc $e(p_i, p_j)$ la valeur de la cellule $Aff(i,j)$. L'algorithme détecte et extrait un ensemble de cycles C où chaque $c_t \in C$ où $t = 1..x$ avec x le nombre de cycles, regroupe les prédicats de sélection qui partagent des valeurs dans Aff.

   Pour cela, l'algorithme sélectionne tout d'abord un nœud p_1 du graphe d'une manière aléatoire. Ce nœud représente le nœud de départ du cycle c_1. Il essaye ensuite d'étendre le cycle c_1 en ajoutant d'autres nœuds. Une fois le cycle formé, ces nœuds sont écartés du graphe, et la même procédure est réitérée pour le reste des nœuds.

   Exemple. La figure 4.6 montre un exemple de graphe G_{Aff_S} construit à partir de la matrice Aff_S de la table 4.3. $C = \{c_1, c_2, ..., c_z\}$, où z represente le nombre de cycles. $c_1 = \{p_1, p_3, p_5\}$.

4. **Composition des termes de prédicats**.

[1] Arbre associé à un graphe contenant tous les nœuds de ce graphe.

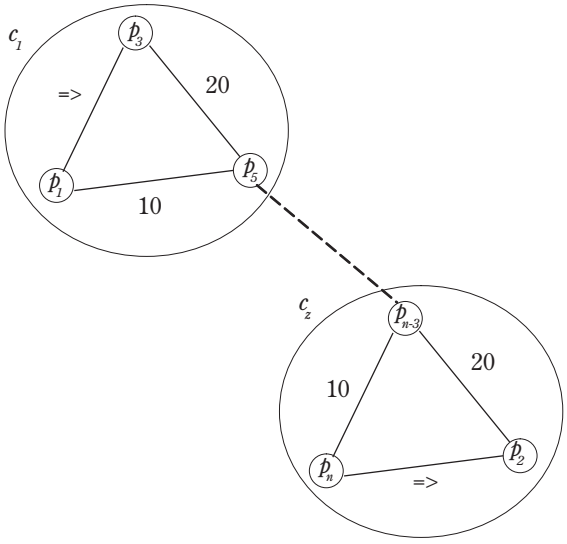

FIG. 4.6 – Exemple de regroupement de prédicats : G_{Aff_S}

Dans cette étape, l'ensemble C est évalué pour déterminer les attributs distincts communs à tous les cycles de C. Ces attributs représentent les attributs de fragmentation (Bellatreche, 2000). Nous construisons ensuite la matrice *Att* qui informe de l'usage d'un attribut par les prédicats d'une classe. A partir de *Att*, nous générons ensuite des termes de prédicats. Un terme de prédicat t est constitué d'un ensemble de prédicats qui couvre tous les attributs de fragmentation. Il représente une entrée dans la matrice *Att* qui peut être étendue en cas d'absence d'un ou plusieurs attributs.

Les termes de prédicats garantissent que les fragments sont disjoints et ne présentent pas une trop grande granularité dans les fragments. Nous illustrons le processus de construction des termes de prédicats par l'exemple suivant.

Exemple. Le tableau 4.4 fournit un exemple de matrice d'usage d'attributs Att_S. Les prédicats du cycle c_1 n'incluent pas l'attribut a_2. c_1 est donc étendu en un cycle c_{11} tel que c_{11} contient des prédicats de c_1 et un prédicat qui inclut l'attribut a_2. $t_1 = p_1 \wedge p_3 \wedge p_5 \wedge p_2$.

5. **Fragmentation des graphes candidats**.

   Chaque terme de prédicat ainsi que qu'un prédicat additionnel, nommé ELSE, constituent un fragment horizontal. Le prédicat ELSE est la négation de la conjonction des prédicats du terme. Il est ajouté pour assurer la complétude.

   Exemple. $t_1 = p_1 \wedge p_3 \wedge p_5 \wedge p_2$ et ELSE=$\neg p_1$ or $\neg p_3$ or $\neg p_5$ or $\neg p_2$ sont des

	a_1	a_2	...	a_r
c_1	1	0		1
c_2	1	1		1
...	0	0		1
c_z	1	1		1

r représente le nombre d'attributs de fragmentation et z le nombre de cycles dans C.

TAB. 4.4 – Exemple de matrice d'usage des attributs : Att_S

termes de prédicat utilisés pour fragmenter le graphe $G_{dimension_{customer}}$.

4.5 Fragmentation basée sur la classification des prédicats (KM)

Dans cette section, nous présentons notre méthode de fragmentation basée sur la technique de classification k-means (Pham et al., 2004). Cette méthode vise à obtenir des fragments qui répondent à des requêtes présentant des caractéristiques communes.

4.5.1 Principe

Cette méthode exploite des prédicats de selection extraits de la charge W et les classifie par la technique des k-means. KM fournit directement un schéma de fragmentation ainsi que les fragments de l'entrepôt de données XML. Elle se déroule en trois étapes.

1. *Codage des prédicats de sélection de W*. Cette étape code les prédicats de P dans une matrice binaire qui représente le contexte de classification.
2. *Classification des prédicats*. Dans cette étape, nous appliquons la technique des k-means pour la classification des prédicats.
3. *Construction des fragments*. Dans cette étape, nous construisons le schéma de fragmentation et nous générons les fragments de l'entrepôts de données XML.

Nous détaillons ces étapes dans les sections 4.5.2, 4.5.3 et 4.5.4, respectivement.

4.5.2 Codage des prédicats de sélection

Dans cette étape, les prédicats de sélection de l'ensemble P (section 4.2.3) sont codés dans une matrice requêtes-prédicats QP. Cette dernière constitue notre contexte

de classification. Son terme général QP_{ij} est égal à un si le prédicat $p_j \in P$ apparaît dans la requête $q_i \in W$ et à zero sinon.

Exemple. La matrice QP_S qui correspond à la charge W_S (section 4.2.3) est représentée par le tableau 4.5.

	p_1	p_2	p_3	p_4	...
q_1	1	0	0	0	
q_2	0	1	1	0	
...					
q_{10}	0	0	1	1	

TAB. 4.5 – Exemple de matrice requêtes-prédicats : QP_S

4.5.3 Classification des prédicats

Les fragments horizontaux sont construits à partir des prédicats $p_i \in P$ ($i = 1..n$ avec n le nombre de prédicats). Nous proposons dans cette étape de regrouper en classes les prédicats présentant des similarités au niveau syntaxique. Pour cela, nous adoptons l'algorithme de classification k-means (Pham et al., 2004). Cet algorithme permet de fixer le nombre de classes et donc de fragments. Notre choix se justifie aussi par la simplicité et les faibles exigences en mémoire de cet algorithme.

Étant donné un entier k, k-means partitionne un ensemble de données en k classes disjointes. Ce résultat est obtenu en positionnant k centroïdes dans l'espace des données sources (les prédicats dans notre cas) et en minimisant la somme des distances intra-classes $\sum_{i=1}^{k} \sum_{x_j \in C_i} (x_j - \mu_i)^2$, où $C_i, i = 1, ..., k$ sont les k classes en sortie et μ_i est le centroïde des points $x_j \in C_i$. Nous notons par $\mathcal{C}$ l'ensemble des classes C_i.

En pratique, nous utilisons la classe Java *SimpleKMeans* du logiciel Weka (Holmes et al., 1994). Cette classe utilise une distance euclidienne pour calculer les distances entre les points des classes. *SimpleKMeans* prend en entrée la matrice QP (ensemble de vecteurs de prédicats p_j) et le paramètre k, et fournit en sortie l'ensemble de classes de prédicats $\mathcal{C}$.

Exemple. Pour la matrice QP_S et avec $k = 2$, SimpleKMeans produit en sortie $C_S = \{\{p_1\}, \{p_2, p_3, p_4\}\}$.

4.5.4 Construction des fragments

L'étape de construction des fragments est constituée de deux sous-étapes : une étape qui construit le schéma de fragmentation de l'entrepôt et une autre qui génère les fragments grâce à un script XQuery. Ces étapes sont présentées dans la figure 4.7.

1. **Construction du schéma de fragmentation**

   Cette étape exploite l'ensemble de classes $\mathcal{C}$ et le document XML qui représente le schéma de l'entrepôt (*dw-model.xml*) pour construire le document $frag_{schema}.xml$ qui représente le schéma de fragmentation.

   Chaque classe de $\mathcal{C}$ représente un fragment et est constituée d'un ensemble de prédicats. Dans le document $frag_{schema}.xml$, elle correspond à un élément $fragment$. Nous identifions par la suite, et pour chaque classe, les dimensions conformes aux prédicats à partir du document *dw-model.xml*. Leurs noms sont indiqués par les éléments *dimension* et les prédicats qui leur correspondent par les éléments *predicate*.

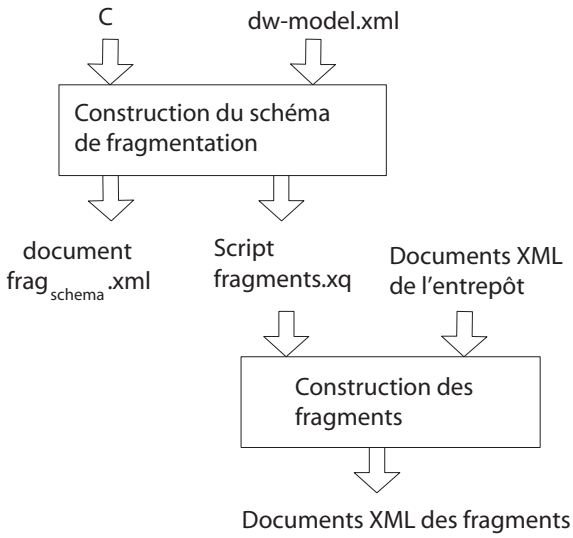

FIG. 4.7 – Étapes de construction des fragments (KM)

Exemple. Le schéma de fragmentation $frag_{schema_S}.xml$ qui correspond à l'ensemble de classes C_S (section 4.5.3) est présenté dans la figure 4.8.

2. **Génération des fragments**

   Cette étape fournit en sortie un ensemble de requêtes XQuery sous la forme d'un script *fragments.xq*. Ce script est construit à partir du graphe G_{schema}. Il est constitué de $|D|$ (avec $|D|$ le nombre de dimensions) requêtes qui génèrent

```
<Schema>
     <fragment id="f1">
          <dimension name="Customer">
               <predicate name="p1" />
          </dimension>
     </fragment>
     <fragment id="f2">
          <dimension name="Customer">
               <predicate name="p2" />
          </dimension>
          <dimension name="Part">
               <predicate name="p3" />
          </dimension>
          <dimension name="Date">
               <predicate name="p4" />
          </dimension>
     </fragment>
</Schema>
```

FIG. 4.8 – Exemple de document de schéma de fragmentation : *frag-schema$_S$.xml*

les fragments des dimensions et d'une requête qui construit le fragment faits. Les requêtes qui construisent les dimensions utilisent les prédicats de sélection des nœuds *predicate*.

Le script *fragments.xq* est exécuté sur l'entrepôt de données XML, c'est-à-dire sur l'ensemble des documents $facts_f.xml$ et $dimension_d.xml$. Il retourne un ensemble de documents $facts_{f_i}.xml$ et $dimension_{fragment_i}.xml$ $(i = 1, ..., k)$ qui représentent les fragments de l'entrepôt XML.

Exemple. La figure 4.9 fournit un exemple de script *fragments$_s$.xq* qui permet de construire le fragment $f2$ de la figure 4.8.

Ce script construit tout d'abord les fragments des dimensions. Pour cela, il exploite des requêtes contenant des prédicats de sélection et les applique sur chaque dimension (les trois premières requêtes de la figure 4.9). Les faits sont ensuite dérivés par une requête (la dernière requête de la figure 4.9) qui exécute une opération de jointure entre le document $fact_f.xml$ et les fragments des dimensions.

3. **Génération du fragment ELSE**

   Nous construisons finalement un $(k+1)^{ème}$ fragment basé sur un prédicat $ELSE$. Le fragment $ELSE$ est la négation des conjonctions de tous les prédicats de P. Il permet de garantir le critère de complétude (Section 2.2.1).

```
element dimension{ attribute dim-
id{Customer}, element Level{
attribute id {Customers},
for $x in document("dimension_Customer.xml")//Level
where $x//attribute[@id="c_nation_key"]/@value="13"]
return $x }
}
element dimension{ attribute dim-
id{Part}, element Level{
attribute id {Part},
for $x in document("dimension_Part.xml")//Level
where $x//attribute[@id="p_type"]/@value="PBC"]
return $x }
}
element dimension{ attribute dim-
id{Date}, element Level{
attribute id {Date},
for $x in document("dimension_Date.xml")//Level
where $x//attribute[@id="d_date_name"]/@value="Sat."]
return $x }
}
element FactDoc {
for $x in //FactDoc/Fact,
 $y in document("dimension_Customer_f2.xml")//instance,
 $z in document("dimension_Part_f2.xml")//instance,
 $t in document("dimension_Date_f2.xml")//instance
where $x/dimension[@dim-
id="Customer"]/@value-id=$y/@id
and $x/dimension[@dim-
id="Part"]/@value-id=$z/@id
and $x/dimension[@dim-
id="Date"]/@value-id=$t/@id
return $x
}
```

FIG. 4.9 – Exemple de script de construction de fragment : $fragments_S.xq$

Exemple. Pour l'ensemble de classes C_S (section 4.5.3), $ELSE = \neg(p_1 \wedge p_2 \wedge p_3 \wedge p_4)$.

4.6 Bilan

Nous avons proposé dans ce chapitre des méthodes de fragmentation pour les entrepôts de données XML. Ces méthodes se basent sur une charge de requêtes XQuery pour dériver un schéma de fragmentation.

Nous avons tout d'abord proposé l'adaptation des méthodes de fragmentation horizontale les plus efficaces dans le contexte relationnel aux entrepôts de données XML. A notre connaissance, ces adaptations constituent les premières contributions dans le contexte des entrepôts de données XML.

Nous avons par la suite proposé une méthode de fragmentation originale basée sur une technique de fouille de données. Cette méthode utilise l'algorithme de classification k-means et permet de contrôler le nombre de fragments. A la différence des méthodes précédentes, elle permet de fusionner les deux phases de fragmentation horizontale dérivée (fragmentation primaire et dérivée) en une seule.

Nous avons également évalué et validé ces propositions. Les expériences que nous avons menées dans ce cadre sont détaillées dans le chapitre 6. Ces expériences visent à comparer nos méthodes de fragmentation en utilisant différentes configurations d'entrepôts et de charges de requêtes (taille de l'entrepôt et nombre de requêtes).

Chapitre 5

Répartition des entrepôts de données XML sur grille

5.1 Contexte et motivation

Afin de satisfaire les besoins en volume de stockage et en capacité de traitement d'un entrepôt de données XML, nous proposons dans ce chapitre de le répartir sur une architecture distribuée. Dans ce contexte, plusieurs solutions peuvent être envisagées : architecture client-serveur, pair-à-pair ou grille de données.

Les architectures client-serveur centralisent toutes les données sur une seule machine (le serveur). Un serveur fournit un ensemble de services qui s'exécutent sur d'autres machines (les clients). Ce type d'architectures, qui est typiquement adopté par les SGBD natifs XML, n'est pas adapté à nos besoins, car la charge du système est intégralement portée par le serveur. Les problèmes de volume de données et de performance des requêtes demeurent donc.

Par opposition, les systèmes pair-à-pair sont spécifiquement conçus pour l'échange et le partage de données entre plusieurs utilisateurs. Dans ces systèmes, un pair (site ou nœud de l'architecture) connaît uniquement ses voisins. Ils sont composés d'un module de recherche de ressources, d'un module de transport de données et d'un module d'indexation. Le module de recherche de ressources permet l'identification d'un pair capable de fournir la ressource demandée. Le module de transport assure le transfert de données entre pairs et utilise des protocoles spécifiques (TCP, HTTP...). Le module d'indexation permet de limiter le nombre de messages nécessaires pour la découverte d'un pair.

Les systèmes pair-à-pair sont adoptés dans les contextes des entrepôts de données relationnels et de documents XML. Certains travaux les exploitent pour effectuer des analyses OLAP sur un entrepôt réparti (Kalnis et al., 2002). Pour cela, ils

se basent sur des agents et proposent l'utilisation des caches des pairs pour améliorer les performances du traitement des requêtes. D'autres travaux exploitent les systèmes pair-à-pair pour la publication et l'échange de données XML entres différents sites (Abiteboul et al., 2005). Ces travaux proposent l'utilisation de structures spécifiques (*hash tables*) pour l'identification des ressources.

Finalement, les systèmes pair-à-pair sont des architectures dites complètement décentralisées. En effet, elles ne fournissent aucune description des différents pairs (données stockées, capacité mémoire et de calcul). Un pair ne se base que sur les informations de ses voisins pour rechercher des données. Ces systèmes sont également dotés d'une grande capacité de passage à l'échelle. En effet, des pairs peuvent s'ajouter et se retirer sans toutefois perturber le fonctionnement du système.

Dans notre contexte d'entreposage, cette caractéristique n'est toutefois pas souhaitable car nous devons pouvoir répondre à tout moment à toute requête formulée sur l'entrepôt distribué, ce que l'absence de contrôle sur les pair rend hasardeux.

Pour terminer, les systèmes de grille ont été développés pour les applications scientifiques de calcul ou de stockage massifs. Ces systèmes ont pour ambition d'établir une infrastructure permettant de faire communiquer des ressources de calcul, des instruments de mesure et de grandes bases de données. Ils permettent également de faciliter l'accès et l'organisation de ces ressources pour l'exécution d'applications scientifiques. Ils fournissent des mécanismes de communication et d'échange de données par services. Ils sont spécialement développés pour le stockage de grandes masses de données (grilles de données) et/ou le support d'applications nécessitant de grandes capacités de calculs (grilles de calcul).

Plusieurs travaux proposent de répartir un entrepôt de données sur une grille (section 2.3). Ces travaux avancent des stratégies d'identification des sites qui stockent les fragments de l'entrepôts (Wehrle et al., 2005). Pour cela, ils définissent des structures d'index (Berchtold et al., 1996) ou implantent des modules spécifiques (Costa & Furtado, 2007). L'échange et le traitement des données sont aussi effectués par des services fournis par la grille (Wehrle et al., 2007) afin de minimiser les coûts de communication.

Les grilles supportent également un grand nombre de nœuds (passage à l'échelle), tout en permettant de connaître l'évolution de l'architecture (nouveaux sites, sites hors ligne, capacité de stockage...), contrairement aux architectures pair-à-pair. Les grilles présentent donc une base intéressante pour la mise en place d'applications décisionnelles distribuées.

5.2 Démarche de distribution

Notre démarche suit une stratégie descendante. Elle est constituée de deux étapes principales : la fragmentation, puis l'allocation des fragments sur les nœuds de la grille. La fragmentation exploite une méthode de fragmentation horizontale dérivée (chapitre 4). L'allocation consiste à stocker les fragments obtenus sur les sites de la grille. Cette allocation est faite de façon aléatoire.

Finalement, nous stockons la localisation des fragments dans le document $Repartition_{schema}.xml$ qui représente le schéma de répartition. Un exemple de ce schéma est fourni dans la figure 5.1. Ce document est constitué d'éléments *node* qui représentent les nœuds de la grille. Ils sont identifié par l'attribut *@id*. Chaque élément *node* est composé d'éléments *fragment* qui définissent les fragments stockés dans un nœud. Un élément *fragment* contient les prédicats de sélection utilisés pour sa construction.

```
<Schema>
    <node id="host_name_1">
        <fragment id="f_1">
            <dimension name="Customer">
                <predicate name="p1" />
            </dimension>
        </fragment>
    </node>
    <node id="host_name_2">
        <fragment id="f_2">
            <dimension name="Customer">
                <predicate name="p2" />
            </dimension>
            <dimension name="Part">
                <predicate name="p3" />
            </dimension>
            <dimension name="Date">
                <predicate name="p4" />
            </dimension>
        </fragment>
    </node>
</Schema>
```

Fig. 5.1 – Exemple de schéma de répartition

5.3 Interrogation de l'entrepôt sur grille

Le processus d'interrogation est représenté dans la figure 5.2. Ses principaux composants sont organisés en deux niveaux.

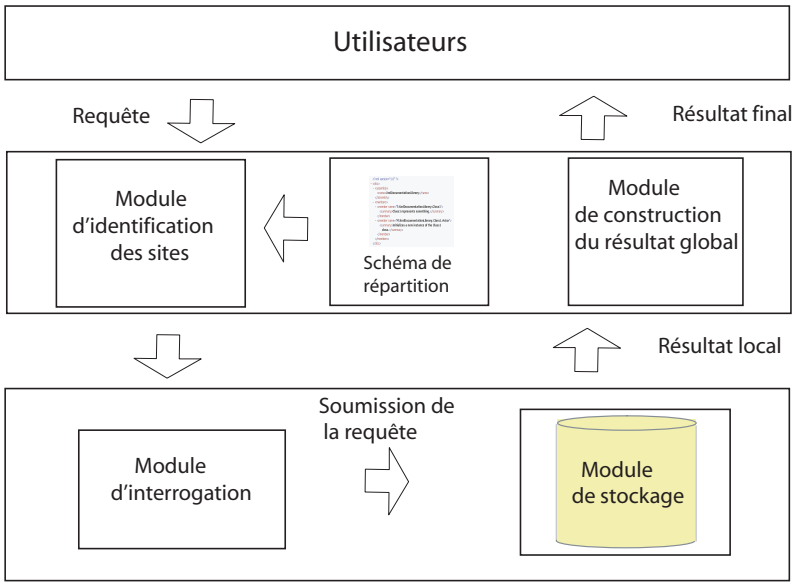

FIG. 5.2 – Interrogation d'un entrepôt XML réparti sur grille

- **Module de stockage**. Ce module est un SGBD natif ou compatible XML qui permet le stockage d'un ou plusieurs fragments sur un site.
- **Module d'interrogation**. Ce module permet la soumission de requêtes sur un site. Il est spécifique au SGBD implanté sur le site.
- **Module d'identification de sites**. Ce module permet l'identification des sites contenant les fragments nécessaires au traitement d'une requête.
  Lors de la soumission d'une requête sur la grille, il identifie les nœuds contenant les fragments nécessaires à son exécution à l'aide du schéma de répartition. Ce processus est constitué des étapes suivantes.

  1. *Analyse de la requête.* Cette étape consiste à extraire les prédicats de sélection de la requête pour construire un ensemble P. Ce processus est effectué par une analyse syntaxique de la requête. Les prédicats de sélection font partie de la clause `Where` de la requêtes.
  2. *Identification des sites.* Dans cette étape, les sites contenant les fragments nécessaires au traitement de la requête sont identifiés. L'identification des sites est effectuée par un parcours des éléments du document $Repartition_{schema}.xml$. Elle identifie tout d'abord les éléments *predicate* dont l'attribut *@name* est égal à un ou plusieurs prédicats de P. A partir

de ces éléments, elle identifie ensuite les éléments *node* qui correspondent aux sites de la grille.

– **Module de construction du résultat global**. Le résultat final de la requête est obtenu par une opération d'union des résultats des sous-requêtes.

5.4 Bilan

Dans cette section, nous avons présenté une première démarche pour la répartition des entrepôts de données XML sur grille. Elle consiste tout d'abord à fragmenter un entrepôt, puis à répartir les fragments obtenus sur les nœuds de la grille. Pour cela, nous avons défini une architecture qui inclut des modules permettant de gérer et d'interroger les données de l'entrepôt réparti. Ces travaux sont préliminaires, mais nous ont toutefois permis de mesurer les temps de traitement d'une charge de requêtes sur un entrepôt réparti sur grille. Nous présentons ces expériences dans le chapitre 6.

Chapitre 6

Validation expérimentale de la fragmentation et de la répartition sur grille

Afin de valider nos contributions, nous avons mené des expériences qui nous ont permis d'évaluer et de comparer les techniques de fragmentation des entrepôts de données XML présentées dans le chapitre 4. Nous avons également évalué les performances d'interrogation d'un entrepôt de données XML sur grille. Ces expériences exploitent un entrepôt de données XML généré par le banc d'essais XWeB (*XML Data Warehouse Benchmark*) et sa charge de requêtes décisionnelles.

6.1 Conditions expérimentales

6.1.1 Banc d'essais XWeB

L'évaluation des performances des SGBD en général, et les tests d'efficacité des techniques d'optimisation des performances en particulier, sont généralement effectués à l'aide de bancs d'essais. Dans le contexte XML, les bancs d'essais existants (Schmidt et al., 2003; Yao et al., 2004) ne sont pas adaptés pour l'évaluation des performances des applications décisionnelles. En effet, ils proposent des bases de données transactionnelles et leurs charges exploitent des requêtes simples. Or, une application décisionnelle se base sur un entrepôt modélisé de façon multidimensionnelle et exploite des requêtes décisionnelles complexes. Nous proposons donc dans cette section un banc d'essais pour entrepôts de données XML : XWeB (*XML Data Warehouse Benchmark*) (Mahboubi & Darmont, 2006).

XWeB est constitué de deux composants principaux (figure 6.1) : un modèle de

base de données (section 6.1.1) qui respecte la représentation que nous avons proposée pour les entrepôts de données XML (section 3.2) et un modèle de charge de requêtes (section 6.1.1). Une évaluation de performance par XWeB est donc effectuée en appliquant la charge de requêtes sur la base de données selon un protocole d'exécution. Ce dernier consiste en un test de chargement de données dans la base et un test de performance (exécuté deux fois). Il est similaire à celui défini dans le banc d'essais TPC-H (TPC, 2005b)

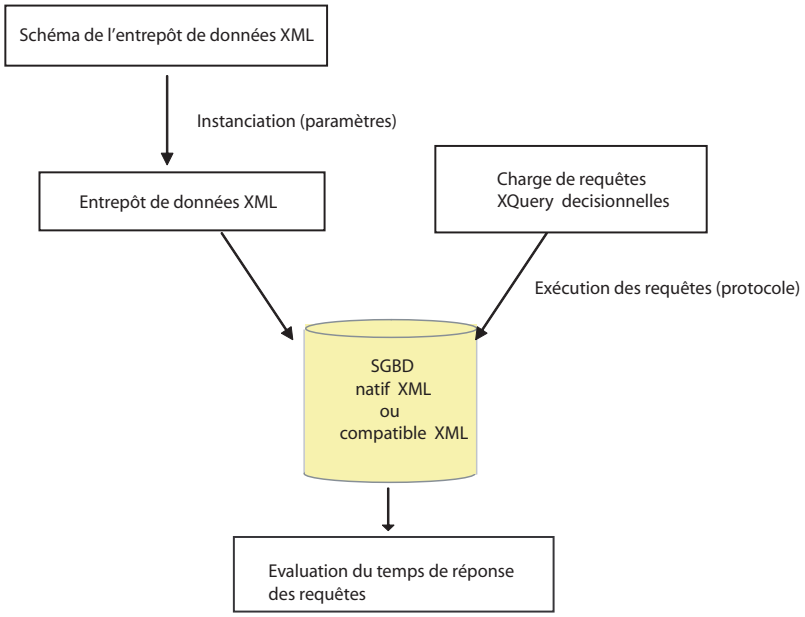

FIG. 6.1 – Principe d'utilisation de XWeB

Entrepôt de données XWeB

Schéma de la base de données. Afin de disposer d'une base reconnue, et comme il n'existe à notre connaissance aucun banc d'essais pour entrepôts de données XML, nous nous sommes appuyés sur le banc d'essais décisionnel relationnel TPC-H (TPC, 2005b) pour concevoir le modèle de base de données de XWeB. Nous avons sélectionné TPC-H pour la simplicité de son modèle, en opposition à celui de TPC-DS (TPC, 2005a), dont la structure en constellation est plus difficile à mettre en oeuvre. De plus, TPC-DS n'est actuellement disponible que sous forme de *draft* et le TPC (*Transaction Performance Processing Council*)[1] qui l'édite ne propose pas encore d'outil logiciel comme ceux qui sont disponibles pour TPC-H.

Nous avons donc étendu TPC-H et remodélisé ses données dans un schéma multidimensionnel en flocon de neige. Le modèle conceptuel est présenté sous la forme d'un

[1] http://www.tpc.org/

diagramme de classes UML dans la figure 6.2. Il représente le cas d'école classique de faits de ventes caractérisés par les mesures quantité de produits achetés (*quantity*) et montant des produits achetés (*amount*). Ces faits sont décrits par les dimensions *produits, consommateurs, fournisseurs* et *temps* qui correspondent à *Dim_Parts*, *Dim_Customers*, *Dim_Suppliers* et *Dim_Date* dans la figure 6.2, respectivement. Selon ce schéma, les *consommateurs* et les *fournisseurs* peuvent être regroupés par *nation* et par *région*. La dimension *temps* présente trois niveaux de hiérarchie : *jour*, *mois* et *année*. Ce schéma est stocké dans un document *dw-model.xml* (section 3.2).

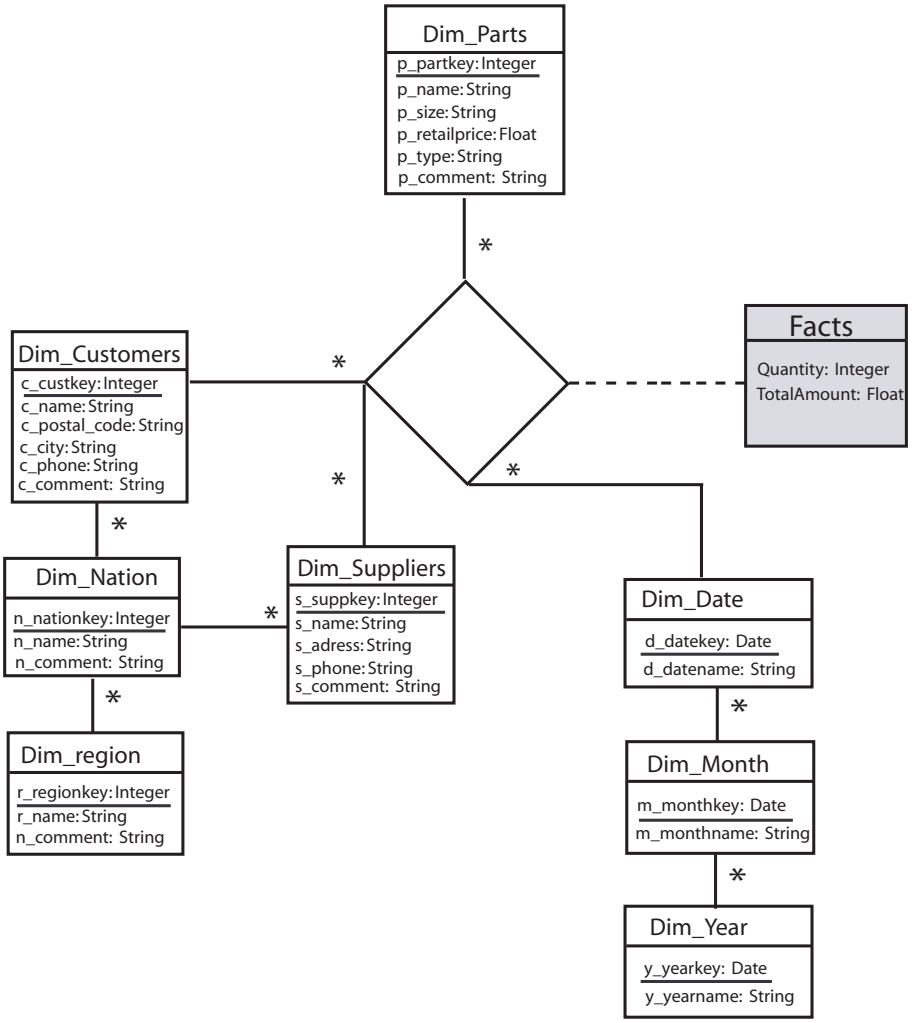

FIG. 6.2 – Schéma conceptuel de l'entrepôt XWeB

Instanciation du schéma. Dans cette phase, nous générons et alimentons les documents XML de l'entrepôt. Pour cela, nous utilisons le module *dbgen* de TPC-H pour générer des données synthétiques. *dbgen* produit des fichiers plats contenant des données textuelles et numériques. Nous utilisons ces fichiers pour alimenter notre

entrepôt de données. L'instanciation du schéma de la base est effectuée en deux étapes. Nous construisons tout d'abord les documents XML des dimensions. Ensuite, nous générons le document des faits.

1. **Construction des documents des dimensions.** Les documents XML des dimensions sont construits sur la base des informations stockées dans le document *dw-model.xml*. En effet, ce document contient la spécification du nombre de dimensions de l'entrepôt, les niveaux hiérarchiques et les attributs de chaque dimension. Ces métadonnées permettent de générer la structure (éléments, attributs et hiérarchies) des documents des dimensions. Les données qui alimentent ces documents XML sont obtenues depuis des fichiers générés par le module *dbgen*.
2. **Construction des documents des faits.** Le processus de construction des faits respecte l'algorithme 2, qui exploite la notion de densité. Cette notion a été introduite dans le banc d'essais DWEB (*Data Warehouse Engineering Benchmark*) (Darmont et al., 2005a) et permet de contrôler le nombre de faits et ainsi la taille du document de faits. Une densité de un indique que toutes les combinaison possibles des instances des dimensions sont présentes dans les faits générés. Les faits obtenus sont stockés dans le document $fact_{sales}.xml$.

Algorithme 2 Algorithme de construction des faits de XWeB

```
1: pour c ∈ Dim_Customers faire
2:   pour p ∈ Dim_Parts faire
3:     pour s ∈ Dim_Suppliers faire
4:       pour d ∈ Dim_Date faire
5:         si Random(0, 1) ≤ Density alors
6:           Quantity = Random(1, 10000)
7:           TotalAmount = Quantity × p.p_retailprice
8:           Create_Fact(p, s, d, Quantity, TotalAmount)
9:         fin si
10:      fin pour
11:    fin pour
12:  fin pour
13: fin pour
```

Paramètres. Les paramètres de la base de données XWeB permettent à l'utilisateur de contrôler la taille de l'entrepôt. La taille S de l'entrepôt est contrôlée par le paramètre SF (*Scale Factor*) hérité de TPC-H. Ce paramètre permet de déterminer la taille de la base de données de TPC-H (de 1 à 100 000 Go). S est estimé comme suit : $S = S_{dimensions} + S_{facts}$, où $S_{dimensions}$ représente la taille des dimensions, qui

ne varie pas quand SF est fixé, et S_{facts} définit la taille des faits, qui dépend de la densité. Les tailles des documents XML des dimensions et des faits peuvent être estimées par les formules suivantes.

$$S_{dimension} = \sum_{d \in D} |d|_{SF} \times nodesize(d)$$

et

$$S_{facts} = \prod_{d \in D} |d|_{SF} \times density \times cellsize$$

D est l'ensemble des dimensions, $|d|_{SF}$ le nombre d'instances d'une dimension d en fonction de SF, $nodesize(d)$ la taille physique moyenne d'un nœud *instance* dans un document dimension, et *cellsize* la taille physique moyenne d'un nœud $fact$ du documents de faits. Le tableau 6.1 montre la taille du document $facts_{sales}.xml$ pour $SF = 1$ et $density = 1$. Nous considérons dans cet exemple que la taille physique moyenne d'un nœud est de 220 octets.

|Customers_dim|	|Suppliers_dim|	|Parts_dim|	|Dates_dim|	S_{facts}(Go)
400	400	500	126	2065

TAB. 6.1 – Nombre d'instances des dimensions et taille du document $facts_{sales}.xml$ pour $SF = 1$

Dans nos expériences, nous utilisons un entrepôt de données XWeB dont les caractéristiques sont listées dans le tableau 6.2.

Faits	**Nombre maximum de faits**
Faits ventes	7000
Dimensions	**Nombre d'instances**
Consommateur	1000
Fournisseur	1000
Temps	500
Produit	1000
Documents	**Taille (Mo)**
$facts_{Sales}.xml$	2,14
$dimension_{Customer}.xml$	0,431
$dimension_{Supplier}.xml$	0,485
$dimension_{Date}.xml$	0,104
$dimension_{Part}.xml$	0,388

TAB. 6.2 – Caractéristiques de l'entrepôt de données XML test

Charge de XWeB. La charge du banc d'essais XWeB comporte vingt requêtes XQuery, numérotées $Q1$ à $Q20$. Ces requêtes exploitent la totalité du schéma de l'entrepôt par des opérations de sélection, de jointure et d'agrégation. La spécification de chaque requête de la charge est décrite dans le tableau 6.3. Le code XQuery correspondant est disponible en annexe 4.

Requête	**Specification**
$Q1$	Nombre d'achats par des consommateurs français, groupé par jour
$Q2$	Nombre d'achats par des consommateurs français et anglais, groupé par mois
$Q3$	Nombre d'achats par des consommateurs anglais de produits français
$Q4$	Nombre d'achats des consommateurs anglais, groupé par produit
$Q5$	Nombre d'achats par des consommateurs français de produits asiatiques les samedis
$Q6$	Nombre d'achats des consommateurs lyonnais les samedis
$Q7$	Nombre d'achats des consommateurs lyonnais, groupé par mois
$Q8$	Nombre d'achats du produit "SMALL PLATED BRASS" par des consommateurs parisiens
$Q9$	Nombre d'achats de produits parisiens par des consommateurs lyonnais les samedis
$Q10$	Nombre d'achats du produit "SMALL PLATED BRASS", groupé par année
$Q12$	Nombre d'achats du produit "SMALL PLATED BRASS" livré par des parisiens et acheté par les lyonnais
$Q13$	Nombre d'achats les lundis
$Q14$	Nombre d'achats par des consommateurs parisiens, groupé par mois
$Q16$	Nombre d'achats du produit "SMALL PLATED BRASS" les lundis
$Q17$	Nombre d'achats par des consommateurs lyonnais du produit "SMALL PLATED BRASS" les mercredis
$Q18$	Nombre d'achats des produits le lundi, groupé par consommateur
$Q19$	Nombre d'achats par des consommateurs lyonnais les samedis
$Q20$	Nombre d'achats de produits livrés par des fournisseurs parisiens, groupé par mois

TAB. 6.3 – Spécification de la charge de XWeB

6.1.2 Conditions matérielles et logicielles

Matériel. Nous évaluons nos méthodes de fragmentation sur une machine PC Pentium 2 GHz avec 1 Go de mémoire sous Windows XP. Notre grille de données est composée de quatre machines : deux machines PC Pentium 2 GHz avec 1 Go

de mémoire sous Windows XP et deux autres PC Pentium 2 GHz avec 500 Mo de mémoire sous Linux.

SGBD XML. Nous utilisons le SGBD natif XML X-Hive (X-Hive-Corporation, 2007) pour stocker et interroger les documents XML de l'entrepôt. Ce système organise les documents XML en collections. Il fournit également une API complète qui permet l'accès et l'interrogation des données XML depuis des applications extérieures.

Environnement de grille. Pour mettre en place notre grille, nous avons utilisé Zorilla (Drost et al., 2007; Drost et al., 2006), un prototype de médiateur qui intègre les fonctionnalités requises pour la mise en oeuvre d'applications sur grille. Il permet le transfert de fichiers, l'ordonnancement de tâches et offre des mécanismes de sécurité. Zorilla est facile à installer et ne nécessite pas de configuration spécifique, contrairement à d'autre environnements comme Globus (Globus-Alliance, 2008). C'est cette simplicité qui a dicté notre choix.

Applications. Les approches que nous proposons (fragmentation et répartition), ainsi que le banc d'essais XWeB, sont codés en langage Java. De plus, nous avons développé des interfaces qui permettent l'accès, l'interrogation et la récupération du temps de traitement des requêtes XQuery. Ces interfaces exploitent les API fournies avec le système X-Hive. Nous avons également exploité l'API de Weka pour coder la méthode de fragmentation basée sur les k-means. Nous nous sommes finalement basés sur le toolkit JavaGAT (*Java Grid Application*) (van Nieuwpoort et al., 2007) pour implémenter notre application sur grille. JavaGAT peut être interfacé avec le médiateur Zorilla et fournit un ensemble d'API qui permettent d'accéder aux données des grilles. Il permet la soumission de tâches (des requêtes XQuery dans notre étude) et facilite l'accès aux informations par des services. Les codes sources de nos programmes sont disponibles en ligne[2].

6.2 Expériences concernant la fragmentation

Dans cette section, nous présentons les expériences que nous avons menées pour valider et comparer nos méthodes de fragmentation (CP, BA et KM ; section 4.3, 4.4 et 4.5, respectivement). Ces expériences mesurent les temps de traitement des

[2]http://eric.univ-lyon2.fr/~hmahboubi/source

requêtes de la charge avec et sans fragmentation. Dans un premier temps, nous évaluons les méthodes exploitant une fragmentation horizontale primaire (CP et BA) et analysons la qualité de fragmentation obtenue par ces méthodes quand la taille de l'entrepôt et la charge varient (section 6.2.1). Ces expériences constituent, à notre connaissance, le premier comparatif expérimental des méthodes de fragmentation primaire CP et BA, que ce soit dans le contexte relationnel ou XML. Chronologiquement, nous les avons effectuées avant de proposer notre approche KM.

Ces premières expériences sont menées dans un contexte centralisé. Toutefois, les fragments que nous construisons sont stockés dans des collections distinctes afin de simuler la distribution des données. En effet, chaque collection peut être considérée comme un site/nœud d'une architecture répartie/parallèle et peut être identifiée et interrogée séparément. Pour mesurer le temps de traitement d'une requête, nous identifions dans un premier temps les fragments nécessaires à son exécution. Cette identification est possible à l'aide du graphe G_{Schema} qui représente le schéma de fragmentation (section 4.2.4). Nous exécutons ensuite la requête sur chaque fragment identifié et sauvegardons le temps de traitement. Afin de simuler une exécution parallèle, nous ne prenons en compte que le temps de traitement maximum.

Nous comparons finalement les méthodes CP et BA à notre approche de fragmentation KM basée sur les k-means (section 6.2.2). Nous évaluons également la surcharge engendrée par ces trois méthodes de fragmentation (section 6.2.3).

6.2.1 Gains de performance obtenus par fragmentation

Influence de la taille de l'entrepôt et des caractéristiques de la charge sur la qualité de la fragmentation (CP et BA) Dans ces expériences, nous exploitons trois configurations d'entrepôts XML et de charges de requêtes. Ces configurations sont définies dans le tableau 6.4. Nous y faisons varier la taille de l'entrepôt (c'est-à-dire le nombre de faits) et le nombre de requêtes et de prédicats de sélection de la charge.

	Configuration 1	**Configuration 2**	**Configuration 3**
Entrepôt de données			
Nombre de faits	800	800	4000
Charge de requêtes			
Nombre de requêtes	13	19	19
Nombre d'opérations de jointure	22	35	35
Nombre de prédicats de sélection	20	30	30

Tab. 6.4 – Configurations d'entrepôts et de charges

Les résultats de nos expériences pour les configurations 1, 2 et 3 sont présentés dans les figures 6.3, 6.4 et 6.5, respectivement. Dans ces figures, l'axe des abscisses représente les requêtes de la charge et l'axe des ordonnées le temps d'exécution de chaque requête, avec et sans fragmentation (l'absence de fragmentation est notée SF). Pour la configuration 1, nous obtenons un gain de performance moyen de 72.95 % avec CP et de 76.32 % avec BA par rapport à un entrepôt non fragmenté. Pour la configuration 2, dans laquelle nous augmentons le nombre de requêtes, CP améliore en moyenne le temps d'exécution des requêtes de 74.53 % et BA de 78.32 %. Finalement, dans la configuration 3, nous augmentons le nombre de faits et nous obtenons un gain de performance moyen de 62.59 % avec CP et de 80,17 % avec BA.

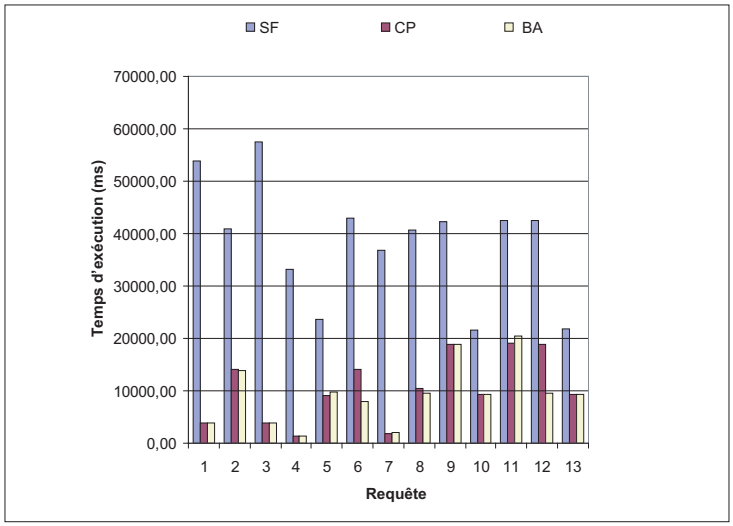

FIG. 6.3 – Temps d'exécution de la charge (configuration 1)

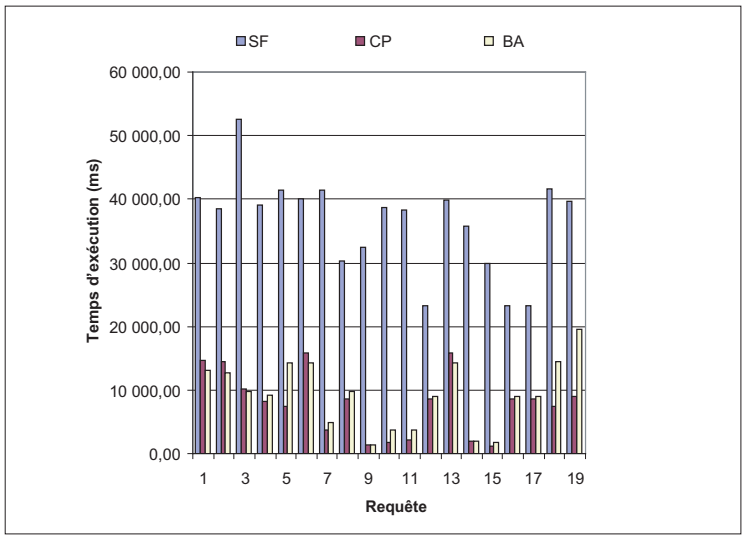

FIG. 6.4 – Temps d'exécution de la charge (configuration 2)

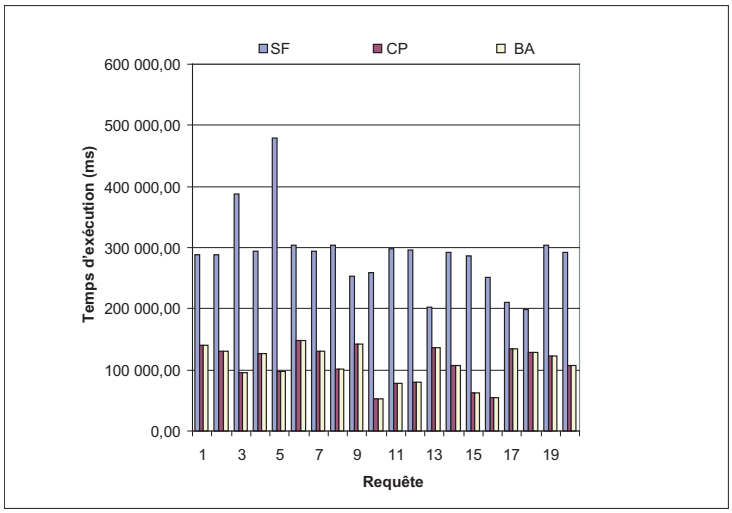

FIG. 6.5 – Temps d'exécution de la charge (configuration 3)

Ces résultats confirment que la fragmentation améliore les performances des requêtes décisionnelles de façon significative. Ils montrent également que BA permet d'obtenir un gain de performance plus élevé que CP dans tous nos tests. Nous pensons que ceci est dû à l'utilisation des fréquences des requêtes par BA, qui favorise le groupement de tous les faits et dimensions nécessaires à une jointure dans un même fragment. De plus, nous remarquons que le gain obtenu avec CP décroît considérablement dans la configuration 3, c'est-à-dire quand la taille de l'entrepôt augmente brusquement. Nous pensons que cette perte d'efficacité est due au nombre de fragments produit par CP, qui est plus grand que celui de BA (159 et 119, respectivement). Les expériences suivantes confirment cette hypothèse.

Influence de la taille de l'entrepôt sur la qualité de la fragmentation Cette série d'expériences permet d'observer l'effet de la taille de l'entrepôt sur le gain de performance obtenu par fragmentation de façon plus détaillée. Nous faisons varier la taille de l'entrepôt de 1000 à 7000 faits et nous mesurons les temps de traitement des requêtes de la charge obtenus par les méthodes de fragmentation CP, BA, KM et sans fragmentation (SF).

Les résultats que nous avons obtenus sont présentés dans la figure 6.6. L'axe des abscisses représente le nombre de faits et celui des ordonnées les temps d'exécution des requêtes de la charge obtenus avant et après fragmentation. Le paramètre k de l'algorithme des k-means est pour l'instant arbitrairement fixé à 8. Rappelons que ce paramètre désigne le nombre de classes généré par l'algorithme des k-means.

Ces résultats montrent que nos méthodes de fragmentation (CP, BA et KM) améliorent le traitement des requêtes de façon encore plus significative lorsque la taille de

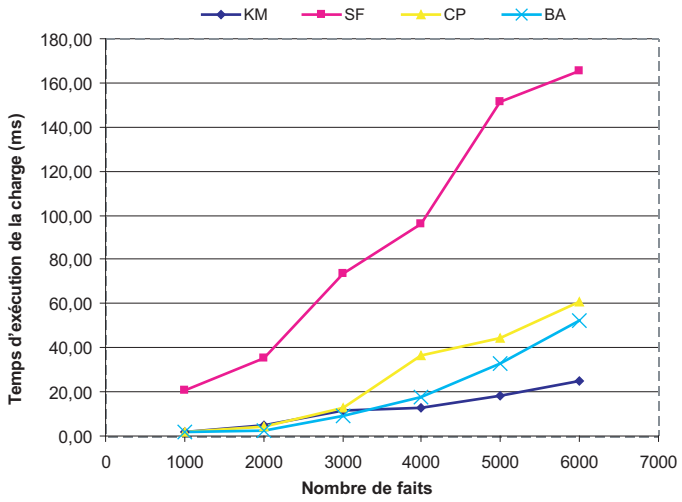

FIG. 6.6 – Comparaison des méthodes de fragmentation

l'entrepôt augmente. Ils montrent également que la méthode de fragmentation KM fournit de meilleures performances que BA et CP. KM accélère en effet le traitement des requêtes en moyenne de 86.5 %, contre environ 36.7 % en moyenne pour BA et KM. Cela est dû au fait que l'approche KM produit moins de fragments (159 avec CP, 119 avec BA et 9 avec KM). En effet, lors du traitement d'une requête sur un entrepôt fragmenté par les méthodes CP ou BA, un nombre important de fragments est sollicité (supérieur à 50 d'après nos observations), ce qui engendre un coût élevé de distribution et de reconstruction du résultat. Le nombre de fragments accédés est nettement moins important avec KM (deux fragments lors de nos expériences).

Nous remarquons aussi que BA présente de meilleures performances que CP quand le nombre de faits augmente. Nous pensons que cela est dû au fait que BA construit des fragments qui contiennent les données requises pour le traitement des opérations de jointure des requêtes les plus fréquentes et enregistre les données accédées par des requêtes peu fréquentes dans un fragment ELSE. CP ne prend pas cet aspect en compte et groupe dans un même fragment des données accédées simultanément par une ou plusieurs requêtes. CP utilise aussi des mintermes qui distribuent les données nécessaires au traitement d'une requête sur plusieurs fragments, ce qui augmente le coût de reconstruction du résultat final.

Les résultats de ces expériences montrent finalement que le gain de performance obtenu par fragmentation décroît quand la taille de l'entrepôt augmente, quelle que soit la méthode employée. Ceci était prévisible. En effet, avec l'augmentation du nombre de faits, les fragments deviennent plus volumineux, ce qui engendre un coût de parcours élevé lors de l'exécution des opérations de jointure.

6.2.2 Influence du nombre de fragments sur les performances

Dans ces expériences, nous observons l'influence du nombre de fragments généré par KM sur le temps de traitement des requêtes de la charge. Pour cela, nous fixons la taille de l'entrepôt (à 4000 et 5000 faits, respectivement) et nous faisons varier le paramètre k.

Les résultats que nous avons obtenus sont présentés dans la figure 6.7. L'axe des abscisses représente les valeurs de k. Notons que $k = 1$ correspond à un entrepôt non fragmenté (pour référence). L'axe des ordonnées représente le temps de traitement des requêtes de la charge. Ces résultats confirment nos observations de la section 6.2.1. Ils montrent en effet que les performances se dégradent dès que le nombre de fragments (classes) augmente. Nous remarquons aussi que le nombre de classes optimal se situe entre 4 et 6 fragments dans nos tests. Il apparaît donc inutile de fixer k à une valeur trop importante.

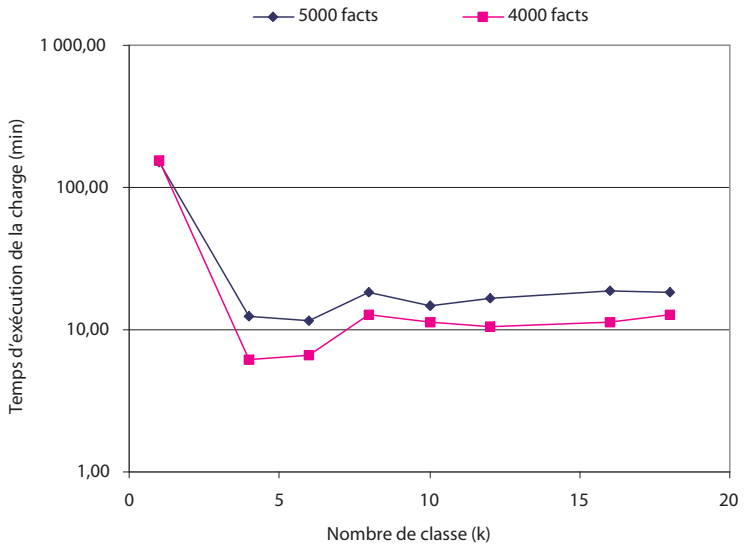

FIG. 6.7 – Influence du nombre de fragments sur la qualité de la fragmentation

6.2.3 Surcharge des méthodes de fragmentation

Nous comparons dans cette section la surcharge des méthodes de fragmentation CP, BA et KM ($k = 8$), c'est-à-dire leur temps d'exécution. Ces expériences visent à trouver un compromis entre les gains obtenus par fragmentation et la surcharge engendrée par les algorithmes de fragmentation.

L'algorithme basé sur les affinités utilise des matrices d'usage (matrice $m \times |P|$ avec m le nombre de requête et $|P|$ le nombre de prédicats de sélection) et d'af-

finités (matrice $|P| \times |P|$). Les complexités de construction de ces matrices sont donc de $O(m \times |P|)$ et de $O(|P|^2)$, respectivement. L'algorithme applique ensuite un regroupement de complexité $O(|P|^2)$. La complexité totale de cet algorithme est donc $O(|P|^2)$. L'algorithme basé sur la construction des prédicats a une complexité de $O(2^{|P|})$ (Bellatreche, 2000). Finalement, l'algorithme KM possède la même complexité que celle de l'algorithme des k-means : $O(|P|)$ (Pham et al., 2004).

Le tableau 6.5 présente les résultats que nous avons obtenus pour un entrepôt de 3000 faits (cette taille est choisie d'une manière arbitraire). Ces résultats expérimentaux montrent que KM s'exécute dans un temps nettement plus faible que CP et BA. On ne retrouve ici pas exactement les complexités des algorithmes des méthodes. Cela est dû aux temps d'exécution des opérations de jointure nécessaires à la construction des fragments de faits. En effet, une requête de construction des faits utilise $|D|$ (avec $|D|$ le nombre de dimensions) opérations de jointure entre les faits et les dimensions de l'entrepôt, ce qui augmente le temps d'exécution des algorithmes de fragmentation. Ces observations nous amènent à dire que la méthode KM peut être envisagée quand l'entrepôt est en phase d'exploitation, tandis que AB et PC doivent être appliquées hors-ligne.

	PC	**AB**	**KM**
Temps d'exécution (h)	16,8	11,9	0,25

TAB. 6.5 – Comparaison des surcharges des méthodes de fragmentation

6.3 Expériences concernant la répartition sur grille

Dans cette section, nous présentons les expériences que nous avons menées pour valider notre démarche de répartition sur grille. Nous avons utilisé la méthode de fragmentation basée sur les k-means pour construire les fragments à répartir. Cette dernière nous permet de générer (avec le paramètre k) le nombre de fragments souhaité en fonction du nombre de machines (nœuds) de la grille. Nous stockons ensuite chaque fragment sur un nœud distinct.

Ces expériences visent, dans un premier temps, à comparer les temps d'exécution des requêtes de la charge avec et sans répartition sur grille (RG et SR, respectivement), et en simulant une répartition par la méthode KM dans un environnement centralisé (section 6.2.1) . Nous présentons ces expériences dans la section 6.3.1. Nous observons également l'influence du nombre de nœuds de la grille sur les performances du traitement des requêtes (section 6.3.2).

6.3.1 Gains de performance obtenus par répartition sur grille

Dans ces expériences, nous mesurons le temps de traitement des requêtes de la charge, en faisant varier la taille de l'entrepôt (de 1000 à 5000 faits). Les résultats que nous avons obtenus sont présentés dans la figure 6.8. L'axe des abscisses représente le nombre de faits et celui des ordonnées les temps d'exécution des requêtes de la charge obtenus avec RG, SR et KM, respectivement. Le nombre de nœuds de la grille est pour l'instant arbitrairement fixé à trois.

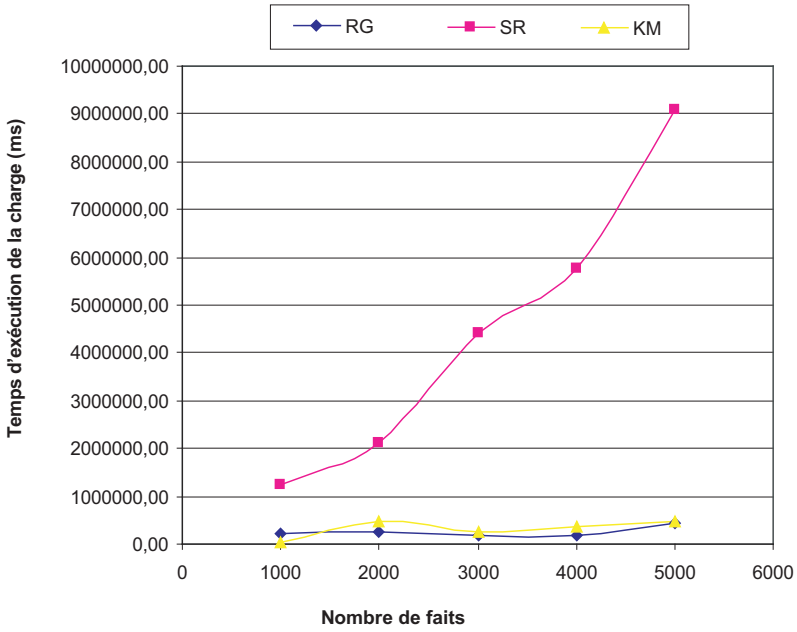

FIG. 6.8 – Temps d'exécution de la charge sur la grille

Ces résultats confirment que la répartition sur grille améliore le temps de traitement des requêtes décisionnelles de façon significative, surtout quand la taille de l'entrepôt est grande. En effet, nous obtenons un gain de performance d'environ 94,18 % en moyenne avec RG par rapport à un entrepôt non réparti (SR). Cette amélioration est due à la distribution du volume de données sur les différents nœuds de la grille, ce qui réduit le nombre d'élément XML parcourus sur un nœud donné pour le traitement d'une requête décisionnelle.

Nos résultats montrent également que les temps moyens de traitement des requêtes obtenus par une répartition simulée avec KM (environnement centralisé) sont proches des résultats réels (répartition sur grille, RG). Toutefois RG présente un temps de traitement des requêtes inférieur de 18,37 % en moyenne à celui obtenu avec KM (même architecture, mais simulée). Nous pensons que cette différence est due aux mécanismes de la grille pour le traitement et la récupération résultats des requêtes. En effet, les services de la grille fusionnent de manière automatique les ré-

sultats des sous-requêtes, tandis que la fusion se fait après récupération des résultats des sous-requêtes dans un environnement centralisé.

6.3.2 Influence du nombre de nœuds sur les performances

Dans ces expériences, nous observons l'influence du nombre de nœuds de la grille sur le temps de traitement des requêtes de la charge. Pour cela, nous faisons varier le nombre de sites (de 2 à 4) et fixons arbitrairement la taille de l'entrepôt à 5000 faits. Les résultats que nous avons obtenus sont présentés dans la figure 6.9. Dans cette figure, l'axe des abscisses représente le nombre de nœuds de la grille et l'axe des ordonnées représente le temps de traitement des requêtes de la charge.

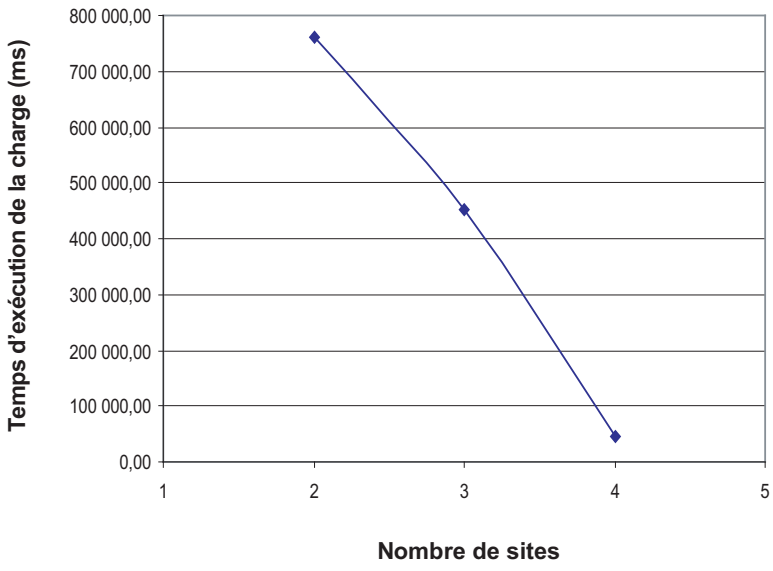

FIG. 6.9 – Influence du nombre de nœuds sur les performance d'execution de la charge

Ces résultats montrent clairement que le temps de traitement de la charge est inversement proportionnel au nombres de nœuds de la grille. On peut donc s'attendre à obtenir de bons temps de réponse en mettant les moyens en terme de nombre de machines. Toutefois, nos propres ressources en ordinateurs ne nous ont pas permis de déterminer quand l'impact de l'augmentation de la taille de la grille s'atténue, voire disparaît.

6.4 Bilan

Dans cette section, nous avons présenté des expériences pour la validation de nos méthodes de fragmentation et de répartition.

Pour cela, nous avons tout d'abord comparé les méthodes de fragmentation horizontale dérivée classiques en faisant varier le nombre de faits et la complexité de la charge. A notre connaissance, cette étude expérimentale constitue le premier comparatif entre les méthodes de fragmentation horizontale classiques CP et BA. Les résultats que nous avons obtenus montrent que la méthode BA permet de meilleures performances que CP lors du traitement des requêtes décisionnelles.

Nous avons par la suite évalué notre méthode de fragmentation basée sur les k-means. Avec cette méthode, nous obtenons un gain de performance plus élevé que celui obtenu par BA et CP. Nous avons finalement comparé les temps d'exécution des algorithmes de fragmentation. Le temps d'execution de l'algorithme KM est également nettement meilleur que ceux de BA et CP, ce qui permet d'envisager son utilisation en ligne.

Nous avons finalement présenté les expériences que nous avons menées pour le traitement de requêtes sur grille. Ces résultats préliminaires montrent que la répartition d'un entrepôt XML sur grille permet un gain de performance considérable lors du traitement des requêtes décisionnelles.

Chapitre 7

Conclusion et perspectives

7.1 Bilan général

Les travaux présentés dans ce mémoire ont été menés au sein du laboratoire ERIC, dont l'activité de recherche est principalement dédiée à l'Extraction de Connaissances à partir des Données (ECD). Ils relèvent plus particulièrement du pôle Bases de Données Décisionnelles (BDD), dont l'une des thématiques de recherche traite de la conception de solutions pour l'entreposage et l'exploitation de données complexes représentées en XML, par l'analyse en ligne (OLAP) et/ou la fouille de données. Dans ce domaine, nous nous sommes plus particulièrement intéressés au problème de la performance des entrepôts de données XML. Nous avons conjointement abordé dans ces travaux la problématique du volume des données et de la performance du traitement de requêtes XML, et proposé une solution par fragmentation, puis par répartition des données de l'entrepôt.

Pour cela, nous nous sommes intéressés dans un premier temps à la fragmentation des entrepôts des données XML et nous avons proposé des techniques qui sont à notre connaissance les premières contributions dans le domaine. Nos méthodes de fragmentation exploitent une charge de requêtes XQuery pour déduire un schéma de fragmentation horizontale dérivée. Nous avons tout d'abord proposé l'adaptation des techniques les plus efficaces du domaine relationnel aux entrepôts de données XML (Mahboubi & Darmont, 2009a), puis une méthode de fragmentation originale basée sur la technique de classification k-means (Mahboubi & Darmont, 2008). Cette dernière nous a permis de contrôler le nombre de fragments. Nous avons finalement proposé une approche de répartition d'un entrepôt de données XML sur une grille. Cette approche est à notre connaissance la première tentative de répartition d'un entrepôt de données XML.

Ces contributions nous ont également amené à proposer un modèle pour les en-

trepôts de données XML (Mahboubi et al., 2009a). Notre modèle unifié étend les modèles existants et propose une représentation de données XML irrégulières. Nous exploitons actuellement ce modèle pour la modélisation d'un entrepôt de données complexes représentées par des documents XML. Cet entrepôt constitue un composant de la plateforme X-WaCoDa (*Warehousing Complex Data*) (Mahboubi et al., 2009b) qui est spécifiquement conçue pour l'entreposage et l'analyse des données complexes (Mahboubi & BenMessaoud, 2006).

Par ailleurs, le développement de nos méthodes inclut une phase de validation expérimentale. Face à l'inexistence d'outils d'évaluation de performance pour les entrepôts XML, nous avons conçu et développé un bancs d'essais décisionnel XML : XWeB (Mahboubi & Darmont, 2006). XWeB nous a permis de comparer les différentes solutions d'optimisation des performances que nous proposons.

Les résultats expérimentaux que nous avons obtenus montrent que notre approche de fragmentation et de répartition sur grille est efficace. Nous avons atteint notre objectif de maîtriser les volumes de données des entrepôts XML et d'améliorer le temps de traitement des requêtes décisionnelles.

7.2 Perspectives de recherche

Les premières expériences que nous avons menées pour valider notre approche de répartition sur grille montrent qu'elle permet de réduire le temps d'exécution de requêtes décisionnelles XQuery de façon significative. Cependant, elles doivent être approfondies. Pour cela, nous envisageons de mener des expériences sur différentes configurations d'entrepôts de tailles variées. Ces expériences nous permettront d'observer l'efficacité de nos méthodes de fragmentation et de répartition sur de grands volumes de données (un million de faits, par exemple). Nous envisageons également d'étendre le nombre de nœuds de la grille pour observer son influence sur les performances des requêtes XML. Notre objectif est d'identifier le nombre de nœuds optimal pour un grand volume de données.

Par ailleurs, en plus de l'optimisation de performance globale par fragmentation et répartition, une optimisation locale de l'interrogation des fragments stockés dans les nœuds de la grille peut être envisagée. Elle permettrait certainement d'améliorer le temps de traitement d'une requête sur un nœud. Pour cela, nous pensons utiliser des structure d'index spécifiques aux entrepôts de données XML (Mahboubi et al., 2006a, 2006b). Watson propose par ailleurs la matérialisation des résultats des requêtes pour accélérer une interrogation identique ultérieure (Watson, 2002). Dans cette optique, nous pourrions utiliser des services pour stocker le résultat d'une

requête dans un nœud approprié.

Finalement, afin d'assurer un fonctionnement efficace de la grille, nous pourrions aussi répliquer les données des fragments sur les différents nœuds de la grille. Dans la littérature, plusieurs techniques de réplication sont proposées (Furtado, 2005), qui pourraient être adaptées au contexte des entrepôts de données XML.

En ce qui concerne la fragmentation, nos résultats expérimentaux montrent que la méthode de fragmentation basée sur les k-means que nous proposons surclasse les adaptations aux entrepôts de données XML des méthodes de fragmentation horizontale dérivée classiques, tant au niveau de l'amélioration des performances des requêtes que de la surcharge engendrée par la fragmentation. Il nous paraît donc intéressant de suivre le chemin inverse de celui adopté dans nos travaux, à savoir d'appliquer notre méthode de fragmentation basée sur les k-means dans les entrepôts de données relationnels.

A un niveau plus global, afin de réduire les fonctions d'administration des systèmes de gestion de bases de données en général et des entrepôts de données en particulier, nous prévoyons de rendre les méthodes que nous proposons les plus automatiques possibles, afin de proposer un outil global d'aide à l'administration des entrepôts de données. L'objectif à terme est de tendre vers l'auto-administration de ces systèmes.

Rendre l'étape de fragmentation de l'entrepôt dynamique grâce à une fragmentation incrémentale pourrait par exemple être envisagé. Cela permettrait de redéfinir le schéma de fragmentation en fonction de l'évolution des besoins d'analyse traduits par les requêtes de la charge. Pour cela, nous pourrions exploiter un algorithme de classification par k-means incrémental (Pham et al., 2004).

De même, Golfarelli *et al.* appliquent une fragmentation verticale sur les vues d'un entrepôts en se basant sur un modèle de coût (Golfarelli et al., 1999). Ils définissent la fragmentation par des opération de décomposition ou de fusion de vues. Ces principes peuvent être envisagés dans le contexte de la fragmentation de données XML, pour faire évoluer la configuration des fragments lors d'une détection de la dégradation des performances.

Un algorithme d'allocation dynamique des fragments sur les nœuds de la grille pourrait également être employé, afin de redéfinir le schéma de répartition en fonction de l'évolution de l'usage des données (Slimani, 2007). Dans le contexte relationnel, cette problématique est traitée par l'utilisation d'algorithmes d'apprentissage (Huang & Chen, 2001) ou par des heuristiques qui utilisent des modèles de coût (Ahmad et al., 2002).

Chapitre 8

Annexes

1 Schéma XML du document *dw-model.xml*

```
<?xml version="1.0" encoding="UTF-8"?> <!-- edited with XMLSpy v2008
sp1 (http://www.altova.com) by Hadj Mahboubi --> <xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="dw-model">
<xs:complexType>
<xs:sequence>
<!-- Fact definition -->
<xs:element name="FactsDoc" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="dimension" maxOccurs="unbounded">
<xs:complexType>
<xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="measure" maxOccurs="unbounded">
<xs:complexType>
<xs:attribute name="name" use="required"/>
<xs:attribute name="type" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
```

```
<xs:attribute name="path" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<!-- Dimension definition -->
<xs:element name="dimensions" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<!--  Hierarchy definition -->
<xs:element name="Level" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:element name="attribute" maxOccurs="unbounded">
<xs:complexType>
<xs:attribute name="name" type="xs:string" use="required"/>
<xs:attribute name="type" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="RollUp" type="xs:string"/>
<xs:attribute name="DrillDown" type="xs:token"/>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="path" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

2 Schéma XML des documents $facts_f.xml$

```
<?xml version="1.0" encoding="ISO-8859-1"?> <!-- edited with XMLSpy
v2008 sp1 (http://www.altova.com) by Hadj Mahboubi --> <xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="FactDoc">
<!-- Fact document root element -->
<xs:complexType>
<xs:sequence>
<xs:element name="Fact" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<!-- Dimension definition -->
<xs:element name="dimension" maxOccurs="unbounded">
<!-- Dimension attributes -->
<xs:complexType>
<xs:attribute name="dim-id" type="xs:string" use="required"/>
<xs:attribute name="value-id" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<!-- Measure definition-->
<xs:element name="measure" maxOccurs="unbounded">
<!-- Measure attributes -->
<xs:complexType>
<xs:attribute name="mes-id" type="xs:string" use="required"/>
<xs:attribute name="value" type="xs:string"  use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
<!-- Fact ID -->
</xs:complexType>
</xs:element>
</xs:schema>
```

3 Schéma XML des documents $dimension_d.xml$

```
<?xml version="1.0" encoding="UTF-8"?> <!-- edited with XMLSpy v2008
sp1 (http://www.altova.com) by Hadj Mahboubi --> <xs:schema
xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
<xs:element name="dimension">
<!-- Dimension root element -->
<xs:complexType>
<xs:sequence>
<!-- Hierarchy levels definition -->
<xs:element name="Level" maxOccurs="unbounded">
<xs:complexType>
 <xs:sequence>
<!-- Hierarchy levels definition -->
<xs:element name="instance" maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<!-- Attribute defitinion  -->
<xs:element name="attribute" maxOccurs="unbounded">
<xs:complexType>
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="value" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
<xs:attribute name="rollUp" type="xs:string" use="required"/>
<xs:attribute name="DrillDown" type="xs:token" use="required"/>
<!--  Hierarchy definition -->
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="id" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attribute name="dim-id" type="xs:string" use="required"/>
```

```
<!-- Dimension ID -->
</xs:complexType>
</xs:element>
</xs:schema>
```

4 Charge de requêtes XQuery de XWeB

```
<!--  Q1 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Customers']/instance where
$a/attribute[@id='c_nationkey']/@value<13 and
$x/dimension[@dim-id='Customers']/@value-id= $a/@id
return $x
,
<!--  Q2 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Customers']/instance where
$a/attribute[@id='c_nationkey']/@value<15 and
$x/dimension[@dim-id='Customers']/@value-id= $a/@id
return $x
,
<!--  Q3 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Customers']/instance, $b in
//dimensions/Level[@id='Suppliers']/instance where
$a/attribute[@id='c_nationkey']/@value>15 and
$b/attribute[@id='s_nationkey']/@value=5 and
$x/dimension[@dim-id='Customers']/@value-id= $a/@id and
$x/dimension[@dim-id='Suppliers']/@value-id= $b/@id
return $x
,
<!--  Q4 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Customers']/instance where
$a/attribute[@id='c_nationkey']/@value>15 and
$x/dimension[@dim-id='Customers']/@value-id= $a/@id
return $x
```

```
,
<!--  Q5 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Customers']/instance, $b in
//dimensions/Level[@id='Suppliers']/instance, $c in
//dimensions/Level[@id='Date']/instance where
$a/attribute[@id='c_nationkey']/@value<13 and
$b/attribute[@id='s_nationkey']/@value>17 and
$c/attribute[@id='d_datename']/@value='Saturday' and
$x/dimension[@dim-id='Customers']/@value-id= $a/@id and
$x/dimension[@dim-id='Suppliers']/@value-id= $b/@id and
$x/dimension[@dim-id='Date']/@value-id= $c/@id
return $x
,
<!--  Q6 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Customers']/instance, $b in
//dimensions/Level[@id='Date']/instance where
$a/attribute[@id='c_nationkey']/@value=13 and
$b/attribute[@id='d_datename']/@value='Saturday' and
$x/dimension[@dim-id='Customers']/@value-id= $a/@id and
$x/dimension[@dim-id='Date']/@value-id= $b/@id
return $x
,
<!--  Q7 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Customers']/instance where
$a/attribute[@id='c_nationkey']/@value=15 and
$x/dimension[@dim-id='Customers']/@value-id= $a/@id
return $x
,
<!--  Q8 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Suppliers']/instance, $b in
//dimensions/Level[@id='Part']/instance where
$a/attribute[@id='s_nationkey']/@value=5 and
$b/attribute[@id='p_type']/@value='SMALL PLATED BRASS' and
```

```
$x/dimension[@dim-id='Suppliers']/@value-id= $a/@id and
$x/dimension[@dim-id='Part']/@value-id= $b/@id
return $x
,
<!--  Q9 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Date']/instance, $b in
//dimensions/Level[@id='Suppliers']/instance, $c in
//dimensions/Level[@id='Customers']/instance where
$a/attribute[@id='d_datename']/@value='Saturday' and
$b/attribute[@id='s_nationkey']/@value=10 and
$c/attribute[@id='c_nationkey']/@value=13 and
$x/dimension[@dim-id='Date']/@value-id= $a/@id and
$x/dimension[@dim-id='Suppliers']/@value-id= $b/@id and
$x/dimension[@dim-id='Customers']/@value-id= $c/@id
return $x
,
<!--  Q10 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Part']/instance where
$a/attribute[@id='p_type']/@value='SMALL PLATED BRASS' and
$x/dimension[@dim-id='Part']/@value-id= $a/@id
return $x
,
<!--  Q11 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Part']/instance, $b in
//dimensions/Level[@id='Suppliers']/instance, $c in
//dimensions/Level[@id='Customers']/instance where
$a/attribute[@id='p_type']/@value='SMALL PLATED BRASS' and
$b/attribute[@id='s_nationkey']/@value=5 and
$c/attribute[@id='c_nationkey']/@value>15 and
$x/dimension[@dim-id='Part']/@value-id= $a/@id and
$x/dimension[@dim-id='Suppliers']/@value-id= $b/@id and
$x/dimension[@dim-id='Customers']/@value-id= $c/@id
return $x
,
```

```
<!--  Q12 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Part']/instance, $b in
//dimensions/Level[@id='Suppliers']/instance where
$a/attribute[@id='p_type']/@value='SMALL PLATED BRASS' and
$b/attribute[@id='s_nationkey']/@value=5 and
$x/dimension[@dim-id='Part']/@value-id= $a/@id and
$x/dimension[@dim-id='Suppliers']/@value-id= $b/@id
return $x
,
<!--  Q13 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Date']/instance where
$a/attribute[@id='d_datename']/@value='Monday' and
$x/dimension[@dim-id='Date']/@value-id= $a/@id
return $x
,
<!--  Q14 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Customers']/instance where
$a/attribute[@id='c_nationkey']/@value=15 and
$x/dimension[@dim-id='Customers']/@value-id= $a/@id
return $x
,
<!--  Q15 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Part']/instance, $b in
//dimensions/Level[@id='Date']/instance where
$a/attribute[@id='p_type']/@value='SMALL PLATED BRASS' and
$b/attribute[@id='d_datename']/@value='Monday' and
$x/dimension[@dim-id='Part']/@value-id= $a/@id and
$x/dimension[@dim-id='Date']/@value-id= $b/@id
return $x
,
<!--  Q16 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Date']/instance, $b in
```

```
//dimensions/Level[@id='Customers']/instance, $c in
//dimensions/Level[@id='Part']/instance where
$a/attribute[@id='d_datename']/@value='Wednesday' and
$b/attribute[@id='c_nationkey']/@value=13 and
$c/attribute[@id='p_type']/@value='SMALL PLATED BRASS' and
$x/dimension[@dim-id='Date']/@value-id= $a/@id and
$x/dimension[@dim-id='Customers']/@value-id= $b/@id and
$x/dimension[@dim-id='Part']/@value-id= $c/@id
return $x
,
<!--  Q17 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Date']/instance where
$a/attribute[@id='d_datename']/@value='Wednesday' and
$x/dimension[@dim-id='Date']/@value-id= $a/@id
return $x
,
<!--  Q18 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Date']/instance where
$a/attribute[@id='d_datename']/@value='Monday' and
$x/dimension[@dim-id='Date']/@value-id= $a/@id
return $x
,
<!--  Q19 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Customers']/instance, $b in
//dimensions/Level[@id='Date']/instance where
$a/attribute[@id='c_nationkey']/@value=13 and
$b/attribute[@id='d_datename']/@value='Saturday' and
$x/dimension[@dim-id='Customers']/@value-id= $a/@id and
$x/dimension[@dim-id='Date']/@value-id= $b/@id
return $x
,
<!--  Q20 -->
for $x in //FactDoc/Fact, $a in
//dimensions/Level[@id='Suppliers']/instance where
```

```
$a/attribute[@id='s_nationkey']/@value=5 and
$x/dimension[@dim-id='Suppliers']/@value-id= $a/@id
return $x
```

5 References

Abiteboul, S., Manolescu, I., & Preda, N. (2005). Constructing and Querying Peer-to-Peer Warehouses of XML Resources. *21st International Conference on Data Engineering (ICDE 05) Tokyo, Japan* (pp. 1122–1123). IEEE Computer Society.

Agrawal, R., & Srikant, R. (1994). Fast Algorithms for Mining Association Rules in Large Databases. *20th International Conference on Very Large Data Bases (VLDB 94), Santiago de Chile, Chile* (pp. 487–499). Morgan Kaufmann.

Ahmad, I., Karlapalem, K., Kwok, Y.-K., & So, S.-K. (2002). Evolutionary Algorithms for Allocating Data in Distributed Database Systems. *Distributed and Parallel Databases*, *11*(1), 5–32.

Andrade, A., Ruberg, G., Baião, F. A., Braganholo, V. P., & Mattoso, M. (2005). *PartiX : processing XQuery Queries over Fragmented XML Repositories.* (Technical report). Computer Science Department, IM Federal University of Rio de Janeiro, Brazil.

Andrade, A., Ruberg, G., Baião, F. A., Braganholo, V. P., & Mattoso, M. (2006). Efficiently Processing XML Queries over Fragmented Repositories with PartiX. *Current Trends in Database Technology, EDBT 2006 Workshops PhD, DataX, IIDB, IIHA, ICSNW, QLQP, PIM, PaRMA, and Reactivity on the Web, Munich, Germany*, Vol. 4254 of *Lecture Notes in Computer Science* (pp. 150–163). Springer.

Aouiche, K., Jouve, P.-E., & Darmont, J. (2006, September). Clustering-Based Materialized View Selection in Data Warehouses. *10th East-European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece*, Vol. 4152 of *Lecture Notes in Computer Science* (pp. 81–95). Springer.

Azefack, S., Aouiche, K., & Darmont, J. (2007). Dynamic index selection in data warehouses. *4th International Conference on Innovations in Information Technology (Innovations 07), Dubai, United Arab Emirates*. IEEE Computer Society.

Baril, X., & Bellahsène, Z. (2003). *Designing and Managing an XML Warehouse*, pp. 455–473. XML Data Management: Native XML and XML-enabled Database Systems. Addison Wesley.

Bellatreche, L. (2000). *Utilisation des Vues Matérialisées, des Index et de la Fragmentation dans la Conception logique et Physique d'un Entrepôt de Données*. PhD thesis, Université de Clermont-Ferrand II.

Bellatreche, L. (2003). Techniques d'optimisation des requêtes dans les data warehouses. *6th International Symposium on Programming and Systems (ISPS 03), Alger, Algeria* (pp. 81–98).

Bellatreche, L., & Boukhalfa, K. (2005). An Evolutionary Approach to Schema Partitioning Selection in a Data Warehouse. *7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 05), Copenhagen, Denmark*, Vol. 3589 of *Lecture Notes in Computer Science* (pp. 115–125). Springer.

Bellatreche, L., Karlapalem, K., & Li, Q. (1998). Derived Horizontal Class Partitioning in OODBs: Design Strategies, Analytical Model and Evaluation. *17th International Conference on Conceptual Modeling (ER 98), Singapore*, Vol. 1507 of *Lecture Notes in Computer Science* (pp. 465–479). Springer.

BenMessaoud, R., Boussaïd, O., & Loudcher-Rabaseda, S. (2006). A Data Mining-Based OLAP Aggregation of Complex Data: Application on XML Documents. *International Journal of Data Warehousing and Mining*, *2*(4), 1–26.

Berchtold, S., Keim, D. A., & Kriegel, H.-P. (1996). The X-tree : An Index Structure for High-Dimensional Data. *22th International Conference on Very Large Data Bases (VLDB 96), Bombay, India* (pp. 28–39). Morgan Kaufmann.

Beyer, K. S., Chamberlin, D. D., Colby, L. S., Ozcan, F., Pirahesh, H., & Xu, Y. (2005). Extending XQuery for Analytics. *ACM SIGMOD International Conference on Management of Data (SIGMOD 05), Baltimore, Maryland* (pp. 503–514). ACM Press.

Beyer, K. S., Cochrane, R., Colby, L. S., Ozcan, F., & Pirahesh, H. (2004). XQuery for Analytics: Challenges and Requirements. *First International Workshop on XQuery Implementation, Experience and Perspectives <XIME-P/>, Paris, France* (pp. 3–8).

Boag, S., Chamberlin, D., Fernández, M., Florescu, D., Robie, J., & Siméon, J. (2004). XQuery 1.0: An XML Query Language. W3C Working Draft, http://www.w3.org/TR/xquery/.

Bonifati, A., & Cuzzocrea, A. (2007). Efficient Fragmentation of Large XML Documents. *18th International Conference on Database and Expert Systems Applications (DEXA 07), Regensburg, Germany*, Vol. 4653 of *Lecture Notes in Computer Science* (pp. 539–550). Springer.

Bonifati, A., Cuzzocrea, A., & Zinno, B. (2006). Fragmenting XML Documents via Structural Constraints. *10th East European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece, Local Proceedings* (pp. 17–29).

Bonifati, A., Matrangolo, U., Cuzzocrea, A., & Jain, M. (2004). XPath lookup queries in P2P networks. *6th ACM CIKM International Workshop on Web Information and Data Management (WIDM 04), Washington, USA* (pp. 48–55). ACM Press.

Bose, S., & Fegaras, L. (2005). XFrag: A query Processing Framework for Fragmented XML Data. *8th International Workshop on the Web and Databases (WebDB 05), Baltimore, Maryland* (pp. 97–102).

Boukraa, D., BenMessaoud, R., & Boussaïd, O. (2006). Proposition d'un Modèle physique pour les entrepôts XML. *Atelier Systèmes Décisionnels (ASD 06), 9th Maghrebian Conference on Information Technologies (MCSEAI 06), Agadir, Morocco.*

Boussaïd, O., BenMessaoud, R., Choquet, R., & Anthoard, S. (2006). X-Warehousing: An XML-Based Approach for Warehousing Complex Data. *10th East-European Conference on Advances in Databases and Information Systems (ADBIS 06), Thessaloniki, Greece*, Vol. 4152 of *Lecture Notes in Computer Science* (pp. 39–54). Springer.

Boussaïd, O., BenMessaoud, R., Choquet, R., & Anthoard, S. (2006). X-Warehousing: an XML-Based Approach for Warehousing Complex Data. *10th East-European Conference on Advances in Databases and Information Systems*

(ADBIS 06), Thessaloniki, Greece, Vol. 4152 of *Lecture Notes in Computer Science* (pp. 39–54). Springer.

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., Yergeau, F., & Cowan, J. (2006). *Extensible Markup Language (XML) 1.1.* W3C, second edition. http://www.w3.org/TR/2006/REC-xml11-20060816/.

Bremer, J.-M., & Gertz, M. (2003). On Distributing XML Repositories. *International Workshop on Web and Databases (WebDB 03), San Diego, California* (pp. 73–78).

Chen, W.-J., Sammartino, A., Goutev, D., Hendricks, F., Komi, I., Wei, M.-P., & Ahuja, R. (2006). *DB2 9 pureXML Guide* (Technical report).

Costa, R. L. C., & Furtado, P. (2007). An SLA-Enabled Grid DataWarehouse. *Eleventh International Database Engineering and Applications Symposium (IDEAS 07), Banff, Alberta, Canada* (pp. 285–289). IEEE Computer Society.

Darabant, A. S., & Campan, A. (2004). Semi-supervised learning techniques: k-means clustering in OODB Fragmentation. *Second IEEE International Conference on Computational Cybernetics (ICCC 04), Vienna, Austria* (pp. 333–338). IEEE Computer Society.

Darmont, J. (2006). Optimisation et évaluation de performance pour l'aide à la conception et à l'administration des entrepôts de données complexes. Mémoire HDR, Université Lumière Lyon 2.

Darmont, J., & Boussaïd, O. (Eds.). (2006). *Managing and Processing Complex Data for Decision Support*. Idea Group Publishing.

Darmont, J., Boussaid, O., & Bentayeb, F. (2005a). DWEB: A Data Warehouse Engineering Benchmark. *7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 05), Copenhagen, Denmark*, Vol. 3589 of *LNCS* (pp. 85–94).

Darmont, J., Boussaïd, O., Ralaivao, J. C., & Aouiche, K. (2005b). An Architecture Framework for Complex Data Warehouses. *7th International Conference on Enterprise Information Systems (ICEIS 05), Miami, USA* (pp. 370–373).

Datta, A., Ramamritham, K., & Thomas, H. M. (1999). Curio: A Novel Solution for Efficient Storage and Indexing in Data Warehouses. *25th International Conference on Very Large Data Bases (VLDB 99), Edinburgh, UK* (pp. 730–733). Morgan Kaufmann.

de Carvalho Costa, R. L., & Furtado, P. (2006). Data Warehouses in Grids with High QoS. *8th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 06), Krakow, Poland*, Vol. 4081 of *Lecture Notes in Computer Science* (pp. 207–217). Springer.

Deshpande, P. M., Ramasamy, K., Shukla, A., & Naughton, J. F. (1998). Caching multidimensional queries using chunks. *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD 98), Seattle, Washington, USA*. ACM Press.

Drost, N., Ogston, E., van Nieuwpoort, R. V., & Bal, H. E. (2007). ARRG: Real-World Gossiping. *16th International Symposium on High-Performance Distributed Computing (HPDC), Monterey, CA, USA* (pp. 147–158). ACM Press.

Drost, N., van Nieuwpoort, R. V., & Bal, H. E. (2006). Simple Locality-Aware Co-

allocation in Peer-to-Peer Supercomputing. *GP2P: Sixth International Workshop on Global and Peer-2-Peer Computing, Singapore.*

Ezeife, C. I., & Barker, K. (1995). A Comprehensive Approach to Horizontal Class Fragmentation in a Distributed Object System. *Distributed and Parallel Databases*, *3*(3), 247–272.

Fiolet, V., & Toursel, B. (2005a). Intelligent Database Distribution on a Grid Using Clustering. *Third International Atlantic Web IntelligenceConference, Advances in Web Intelligence (AWIC 05), Lodz, Poland*, Vol. 3528 of *Lecture Notes in Computer Science* (pp. 466–472). Springer.

Fiolet, V., & Toursel, B. (2005b). Progressive Clustering for Database Distribution on a Grid. *4th International Symposium on Parallel and Distributed Computing (ISPDC 05), Lille, France* (pp. 282–289). IEEE Computer Society.

Fiser, B., Onan, U., Elsayed, I., Brezany, P., & Tjoa, A. M. (2004). On-Line Analytical Processing on Large Databases Managed by Computational Grids. *15th International Workshop on Database and Expert Systems Applications (DEXA 04), Zaragoza, Spain* (pp. 556–560). IEEE Computer Society.

Furtado, P. (2004). Workload-Based Placement and Join Processing in Node-Partitioned Data Warehouses. *6th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 04), Zaragoza, Spain*, Vol. 3181 of *Lecture Notes in Computer Science* (pp. 38–47).

Furtado, P. (2005). Replication on Node Partitioned Data Warehouses. *VLDB Workshop on Design, Implementation, and Deployment of Database Replication, Trondheim, Norway.*

Gertz, M., & Bremer, J.-M. (2003). *Distributed XML Repositories: Top-down Design and Transparent Query Processing.* (Technical report). Departement of Computer Science, University of California, USA.

Globus-Alliance (2008). Globus Toolkit 4.2.1. http://www.globus.org/.

Golfarelli, M., Maio, D., & Rizzi, S. (1999). Vertical Fragmentation of Views in Relational Data Warehouses. *Settimo Convegno Nazionale su Sistemi Evoluti Per Basi Di Dati (SEBD 99), Como, Italy* (pp. 19–33).

Golfarelli, M., & Rizzi, S. (2001). Integrating XML Sources Into a Data Warehouse Environment. *IEEE International Conference on Software, Telecommunications and Computer Networks (SoftCOM 01), Dubrovnik, Croatia* (pp. 49–56).

Golfarelli, M., Rizzi, S., & Vrdoljak, B. (2001). Data Warehouse Design from XML Sources. *4th International Workshop on Data Warehousing and OLAP (DOLAP 01), Atlanta, USA* (pp. 40–47). ACM Press.

Gorawski, M., Gorawski, M., & Bankowski, S. (2007). Selection of Indexing Structures in Grid Data Warehouses with Software Agents. *International Journal of Computer Science & Applications, IJCSA*, *4*(1), 39–52.

Gorla, N., & Betty, P. W. Y. (2008). Vertical Fragmentation in Databases Using Data-Mining Technique. *International Journal of Data Warehousing and Mining*, *4*(3), 35–53.

Gounaris, A., Comito, C., Sakellariou, R., & Talia, D. (2007). A Service-Oriented System to Support Data Integration on Data Grids. *Seventh IEEE International*

Symposium on Cluster Computing and the Grid (CCGrid 07), Rio de Janeiro, Brazil (pp. 627–635). IEEE Computer Society.

Hachicha, M., Mahboubi, H., & Darmont, J. (2007, Octobre). Vers une algèbre XML-OLAP : État de l'art. *2ème Atelier Systèmes Décisionnels (ASD 07), Sousse, Tunisie*.

Hachicha, M., Mahboubi, H., & Darmont, J. (2008). Expressing OLAP operators with the TAX XML algebra. *3rd International Workshop on Database Technologies for Handling XML Information on the Web (DataX-EDBT 08), Nantes, France*.

Holmes, G., Donkin, A., & Witten, I. (1994). Weka: A machine learning workbench. *Second Australia and New Zealand Conference on Intelligent Information Systems, Brisbane, Australia*.

Huang, Y.-F., & Chen, J.-H. (2001). Fragment allocation in distributed database design. *Journal of information science and engineering*, *17*, 491–506.

Hümmer, W., Bauer, A., & Harde, G. (2003). XCube: XML for data warehouses. *6th International Workshop on Data Warehousing and OLAP (DOLAP 03), New Orleans, USA* (pp. 33–40). ACM Press.

Inmon, W. (2005). *Building the data warehouse*. John Wiley & Sons, fourth edition.

Iqbal, S., Bunn, J. J., & Newman, H. B. (2003). Distributed Heterogeneous Relational Data Warehouse In A Grid Environment. *Cumputing in High Energy and Nuclear Physics, La Jolla, USA* (pp. 24–28).

Jagadish, H. V., Al-khalifa, S., Chapman, A., Lakshmanan, L. V. S., Nierman, A., Paparizos, S., Patel, J. M., Srivastava, D., Wiwatwattana, N., Wu, Y., & Yu, C. (2002). TIMBER: A Native XML Database. *International Journal on Very Large DataBase (VLDB J. 02)*, *11*(4), 274–291.

Kalnis, P., Ng, W. S., Ooi, B. C., Papadias, D., & Tan, K.-L. (2002). An adaptive peer-to-peer network for distributed caching of OLAP results. *2002 ACM SIGMOD International Conference on Management of Data (SIGMOD 02), Madison, Wisconsin* (pp. 25–36). ACM Press.

Koreichi, A., & Cun, B. L. (1997). *On Data Fragmentation and Allocation in Distributed Object Oriented Databases* (Technical report). PRiSM, Versailles University, France.

Ma, H., & Schewe, K.-D. (2003). Fragmentation of XML Documents. *XVIII Simpósio Brasileiro de Bancos de Dados, Manaus, Amazonas, Brasil* (pp. 200–214). UFAM.

Ma, H., Schewe, K.-D., Hartmann, S., & Kirchberg, M. (2003). Distribution Design for XML Documents. *3rd International Conference on Electronic Commerce Engineering (IceCE 03), Hangzhou, China* (pp. 1007–1012). International Academic Publisher/World Publishing Corporation.

Ma, H., Schewe, K.-D., & Wang, Q. (2006). A heuristic approach to cost-efficient fragmentation and allocation of complex value databases. *17th Australasian Database Conference on Database Technologies (ADC 06), Hobart, Australia* (pp. 183–192).

Mahboubi, H., Aouiche, K., & Darmont, J. (2006a). Materialized View Selection by Query Clustering in XML Data Warehouses. *4th International Multiconference*

on Computer Science and Information Technology (CSIT 06), Amman, Jordan (pp. 68–77).

Mahboubi, H., Aouiche, K., & Darmont, J. (2006b). Un index de jointure pour les entrepôts de données XML. *6èmes Journées Francophones Extraction et Gestion des Connaissances (EGC 06), Lille*, Vol. E-6 of *Revue des Nouvelles Technologies de l'Information* (pp. 89–94). Toulouse: Cépaduès.

Mahboubi, H., Aouiche, K., & Darmont, J. (2008). A Join Index for XML Data Warehouses. *2008 International Conference on Information Resources Management (Conf-IRM 08), Niagara Falls, Canada*.

Mahboubi, H., & BenMessaoud, R. (2006). Entreposage XML de données complexes. Journée Innovation, Logiciels, Services (ILS), Lyon. Poster.

Mahboubi, H., & Darmont, J. (2006). Benchmarking XML data warehouses. *Atelier Systèmes Décisionnels (ASD 06), 9th Maghrebian Conference on Information Technologies (MCSEAI 06), Agadir, Morocco*.

Mahboubi, H., & Darmont, J. (2008). Data Mining-based Fragmentation of XML Data Warehouses. *ACM 11th International Workshop on Data Warehousing and OLAP (CIKM/DOLAP 08), Napa Valley, USA* (pp. 9–11). ACM Press.

Mahboubi, H., & Darmont, J. (2009a). Enhancing XML Data Warehouse Query Performance by Fragmentation. *24th Annual ACM Symposium on Applied Computing (SAC 09), Hawaii, USA*. ACM Press.

Mahboubi, H., & Darmont, J. (2009b). *Indices in XML databases*. Encyclopedia of Database Technologies and Applications, Second Edition. Idea Group Publishing.

Mahboubi, H., Hachicha, M., & Darmont, J. (2009a). *XML Warehousing and OLAP*, Vol. IV of *Encyclopedia of Data Warehousing and Mining, Second Edition*, pp. 2109–2116. Hershey, PA, USA: IGI Publishing.

Mahboubi, H., Ralaivao, J.-C., Loudcher, S., Boussaïd, O., Bentayeb, F., & Darmont, J. (2009b). *X-wacoda: An xml-based approach for warehousing and analyzing complex data*. Advances in Data Warehousing and Mining. Hershey, PA, USA: IGI Publishing.

Matsuda, H. (2005). A Grid Environment for Data Integration of Scientific Databases. *First International Conference on e-Science and Grid Computing (E-SCIENCE 05), Washington, USA* (pp. 3–4).

Meier, W. (2002). eXist: An Open Source Native XML Database. *Web, Web-Services, and Database Systems, NODe 2002 Web and Database-Related Workshops, Erfurt,Germany*, Vol. 2593 of *Lecture Notes in Computer Science* (pp. 169–183). Springer.

Meier, W. (2005). eXist: An Open Source Native XML Database. http://exist.sourceforge.net/.

Munneke, D., Wahlstrom, K., & Mohania, M. K. (1999). Fragmentation of Multidimensional Databases. *10th Australasian Database Conference (ADC 99), Auckland, New Zealand* (pp. 153–164).

Nassis, V., Rajugan, R., Dillon, T. S., & Rahayu, J. W. (2005). Conceptual and Systematic Design Approach for XML Document Warehouses. *International Journal of Data Warehousing & Mining*, *1*(3), 63–86.

Navathe, S. B., Karlapalem, K., & Ra, M. (1995). A Mixed Fragmentation Methodology for Initial Distributed Database Design. *Journal of Computer and Software Engineering*, *3*(4).

Ng, Y.-K., & Seetharaman, A. (1997). A Query-Based Horizontal Fragmentation Approach for Disjunctive Deductive Databases. *DEXA Workshop* (pp. 604–609).

Niemi, T., Niinimäki, M., & Sivunen, V. (2003). Integrating Distributed Heterogenous Databases and Distributed Grid Computing. *5th International Conference on Enterprise Information Systems (ICEIS 03), Angers, France* (pp. 96–103).

Nieto-Santisteban, M. A., Gray, J., Szalay, A. S., Annis, J., Thakar, A. R., & O'Mullane, W. (2005). When Database Systems Meet the Grid. *Second Biennial Conference on Innovative Data Systems Research (CIDR 05), Asilomar, USA* (pp. 154–161).

Noaman, A. Y., & Barker, K. (1999). A Horizontal Fragmentation Algorithm for the Fact Relation in a Distributed Data Warehouse. *1999 ACM International Conference on Information and Knowledge Management (CIKM 99), Kansas City, USA* (pp. 154–161). ACM Press.

Oracle-Corporation (2006). Oracle9i Data Warehousing Guide Release 2 (9.2). http://download-west.oracle.com/docs/cd/B10501_01/server.920/a96520/toc.htm.

Özsu, M. T., & Valduriez, P. (1999). *Principles of Distributed Database Systems, Second Edition*. Prentice-Hall.

Paparizos, S., Wu, Y., Lakshmanan, L. V. S., & Jagadish, H. V. (2004). Tree Logical Classes for Efficient Evaluation of XQuery. *SIGMOD International Conference on Management of Data (SIGMOD 04), Paris, France* (pp. 71–82).

Park, B.-K., Han, H., & Song, I.-Y. (2005). XML-OLAP: A Multidimensional Analysis Framework for XML Warehouses. *7th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 05)*, Vol. 3589 of *Lecture Notes in Computer Science* (pp. 32–42). Springer.

Pernul, G., Karlapalem, K., & Navathe, S. B. (1991). Relational Database Organization Based on Views and Fragments. *International Conference Database and Expert Systems Applications (DEXA 91), Berlin, Germany* (pp. 380–386). Springer-Verlag.

Pham, D., Dimov, S., & Nguyen, C. (2004). An Incremental K-means algorithm. *Journal of Mechanical Engineering Science*, *218*(7), 783–795.

Pokorný, J. (2002). XML Data Warehouse: Modelling and Querying. *5th International Baltic Conference (BalticDB&IS 06), Tallin, Estonia* (pp. 267–280). Institute of Cybernetics at Tallin Technical University.

Rajugan, R., Chang, E., & Dillon, T. S. (2005). Conceptual Design of an XML FACT Repository for Dispersed XML Document Warehouses and XML Marts. *20th International Conference on Computer and Information Technology (CIT 05), Shanghai, China* (pp. 141–149). IEEE Computer Society.

Ravat, F., & Zurfluh, G. (1995). Issues in the Fragmentation of Object-Oriented Databases. *2nd Basque International Workshop on Information Technology (BIWIT 95), San Sebastian, Spain* (pp. 150–160).

Rusu, L. I., Rahayu, J. W., & Taniar, D. (2005). A Methodology for Building XML Data Warehouse. *International Journal of Data Warehousing and Mining, 1(2)*, 67–92.

Rys, M. (2003). *XQuery in Relational Database Systems*. Addison-Wesley.

Schewe, K.-D. (2002). Fragmentation of Object Oriented and Semi-Structured Data. *Baltic Conference, BalticDB&IS 2002* (pp. 1–14). Kluwer Academic Publishers.

Schmidt, A., Waas, F., Kersten, M. L., Carey, M. J., Manolescu, I., & Busse, R. (2003). Assessing XML Data Management with XMark. *Efficiency and Effectiveness of XML Tools and Techniques and Data Integration over the Web, VLDB 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb*, Vol. 2590 of *LNCS* (pp. 144–145).

Slimani, N. (2007). Entrepôts de données XML répartis sur grille de données. Master's thesis, Université Lyon 2.

Sun-microsystems (2002). API for XML-based RPC (JAX-RPC) 1.0-01. FCS Release Notes.

Torlone, R. (2003). Conceptual Multidimensional Models. In *Multidimensional databases* (pp. 69–90).

TPC (2005a). TPC Benchmark DS (Decision Support) Draft Specification revision 32. Transaction Processing Performance Council. http://www.tpc.org.

TPC (2005b). TPC Benchmark H Standard Specification revision 2.3.0. Transaction Processing Performance Council. http://www.tpc.org.

van Nieuwpoort, R. V., Kielmann, T., & Bal, H. E. (2007). User-Friendly and Reliable Grid Computing Based on Imperfect Middleware. *Proceedings of the ACM/IEEE Conference on Supercomputing (SC'07), Reno, USA* (p. 34). ACM Press.

Vrdoljak, B., Banek, M., & Rizzi, S. (2003). Designing Web Warehouses from XML Schemas. *5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 03), Prague, Czech Republic*, Vol. 2737 of *Lecture Notes in Computer Science* (pp. 89–98). Springer.

W3C (2007). XQuery 1.0 and XPath 2.0 Functions and Operators. W3C Recommendation. http://www.w3.org/TR/xpath-functions/.

Wang, H., Li, J., & Xiong, S. (2005). Efficient Join Algorithms for Integrating XML Data in Grid Environment. *4th International Conference on Grid and Cooperative Computing (GCC 05), Beijing, China*, Vol. 3795 of *Lecture Notes in Computer Science* (pp. 547–553).

Watson, P. (2002). *Databases and the Grid* (Technical report). Version 3.1, Newcasfie University Computing Science Technical Report CS-TR-755.

Watson, P. (2005). Databases in Grid Applications: Locality and Distribution. *22nd British National Conference on Databases, Database: Enterprise, Skills and Innovation, (BNCOD 05), Sunderland, UK*, Vol. 3567 of *Lecture Notes in Computer Science* (pp. 1–16). Springer.

Wehrle, P., Miquel, M., & Tchounikine, A. (2005). A Model for Distributing and Querying a Data Warehouse on a Computing Grid. *11th International Conference on Parallel and Distributed Systems (ICPADS 05), Fuduoka, Japan* (pp. 203–209). IEEE Computer Society.

Wehrle, P., Tchounikine, A., & Miquel, M. (2007). Grid Services for Efficient Decentralized Indexation and Query Execution on Distributed Data Warehouses. *19th International Conference on Advanced Information Systems Engineering (CAiSE'07), Trondheim, Norway* (pp. 13–16).

Wiwatwattana, N., Jagadish, H. V., Lakshmanan, L. V. S., & Srivastava, D. (2007). X^3: A Cube Operator for XML OLAP. *23rd International Conference on Data Engineering (ICDE 07), Istanbul, Turkey* (pp. 916–925). IEEE.

Wu, M.-C., & Buchmann, A. P. (1997). Research Issues in Data Warehousing. *Datenbanksysteme in Buro, Technik und Wissenschaft* (pp. 61–82).

X-Hive-Corporation (2007). The fastest and most scalable XML database powered by open standards. http://www.x-hive.com/products/db/.

Xyleme, L. (2001). Xyleme: A Dynamic Warehouse for XML Data of the Web. *International Database Engineering & Applications Symposium (IDEAS 01), Grenoble, France* (pp. 3–7). IEEE Computer Society.

Yao, B. B., Özsu, M. T., & Khandelwal, N. (2004). XBench Benchmark and Performance Testing of XML DBMSs. *20th International Conference on Data Engineering (ICDE 04), Boston, USA* (pp. 621–633).

Zhang, J., Wang, W., Liu, H., & Zhang, S. (2005). X-warehouse: building query pattern-driven data. *14th international conference on World Wide Web (WWW 05), Chiba, Japan* (pp. 896–897). ACM Press.

Zhang, Y., & Orlowska, O. (1994). On fragmentation approaches for distributed database design. *Information Sciences*, *1*(3), 117–132.