CHARLOTTE DECLERCQ

Conception et développement d'un service web de mise à jour incrémentielle pour les cubes de données spatiales

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en sciences géomatiques
pour l'obtention du grade de Maître ès Sciences (M.Sc.)

FACULTÉ DE FORESTERIE ET DE GÉOMATIQUE
UNIVERSITÉ LAVAL
QUÉBEC

2008

Résumé

Les applications géodécisionnelles évoluent vers le temps réel et nécessitent un mécanisme de mise à jour rapide. Or, ce processus est complexe et très coûteux en temps de calcul à cause de la structure dénormalisée des données, stockées sous forme de cube. La méthode classique qui consistait à reconstruire entièrement le cube de données prend de plus en plus de temps au fur et à mesure que le cube grossit, et n'est plus envisageable. De nouvelles méthodes de mise à jour dites incrémentielles ont fait leurs preuves dans le domaine du Business Intelligence. Malheureusement, de telles méthodes n'ont jamais été transposées en géomatique décisionnelle, car les données géométriques nécessitent des traitements spécifiques et complexes. La mise à jour des cubes de données spatiales soulève des problèmes jusqu'alors inconnus dans les cubes de données classiques. En plus de cela, une large confusion règne autour de la notion de mise à jour dans les entrepôts de données.

On remarque également que l'architecture des entrepôts de données suit la tendance actuelle d'évolution des architectures de systèmes informatiques vers une distribution des tâches et des ressources, au détriment des systèmes centralisés, et vers le développement de systèmes interopérables. Les architectures en émergence, dites orientées services deviennent dans ce sens très populaires. Cependant, les services dédiés à des tâches de mise à jour de cubes sont pour l'heure inexistants, même si ceux-ci représenteraient un apport indéniable pour permettre la prise de décision sur des données toujours à jour et cohérentes.

Le but de ce mémoire est d'élaborer des méthodes de mise à jour incrémentielles pour les cubes spatiaux et d'inscrire le dispositif dans une architecture orientée services. La formulation de typologies pour la gestion de l'entrepôt de données et pour la mise à

jour de cube a servi de base à la réflexion. Les méthodes de mise à jour incrémentielles existantes pour les cubes non spatiaux ont été passées en revue et ont permis d'imaginer de nouvelles méthodes incrémentielles adaptées aux cubes spatiaux. Pour finir, une architecture orientée services a été conçue, elle intègre tous les composants de l'entrepôt de données et contient le service web de mise à jour de cube, qui expose les différentes méthodes proposées.

Remerciements

Cette recherche a été réalisée dans le cadre d'une subvention offerte pour le projet GEOIDE « Un outil web interactif pour mieux comprendre les impacts des changements climatiques sur la santé publique » dirigé par le Dr. Pierre Gosselin de l'INSPQ et le Dr. Thierry Badard. Aussi, je remercie le réseau GEOIDE et le professeur Thierry Badard pour le support financier accordé pour mener cette recherche.

Je remercie le professeur Thierry Badard pour son expertise, son soutien et pour m'avoir fait découvrir et apprécier le domaine OpenSource en géomatique.

Je remercie également mes codirecteurs Jacynthe Pouliot et Yvan Bédard pour leurs précieux conseils et le temps qu'ils ont su m'accorder durant ma maîtrise.

Je voudrais aussi remercier Étienne Dubé pour avoir conçu Geokettle.

Merci à Suzie Larrivée et Éveline Bernier qui ont toujours été présentes pour m'aider et répondre à mes questions.

Je souhaite adresser un grand merci aux étudiants avec qui j'ai partagé les bureaux et qui m'ont supportée durant plusieurs mois, tout particulièrement merci à Eve, Karine, Chloé, Rosemary, Véronique, Leila, Christophe, Claudia et Alejandro.

Je tiens également à remercier le Laboratoire LRTS de l'Université Laval pour m'avoir fourni le template LaTeX modifié utilisé pour rédiger ce mémoire.

Finalement, j'aimerais adresser un merci spécial à Emmanuel, mon conjoint, pour m'avoir encouragée et soutenue moralement durant cette maîtrise et notre séjour au Québec, et à mes parents pour leur support moral et financier, et sans qui je n'aurais pas pu réaliser cette aventure.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 Mise en contexte

Les outils d'aide à la décision se sont multipliés durant ces dernières années avec l'essor d'Internet et de l'informatique. Les organisations collectent de plus en plus de données et les stockent dans des systèmes centralisés à des fins d'analyse. Par exemple les transactions bancaires, les achats au supermarché, la fréquentation des pages web, etc. sont enregistrés dans des systèmes afin d'analyser les habitudes des clients. Ces transactions sont réalisées quasi-instantanément par des systèmes dits « transactionnels ». Chaque transaction engendre un certain nombre d'invocations et de mises à jour de base de données. Si au départ il s'agissait de simples bases de données indépendantes, aujourd'hui on utilise des entrepôts de données d'un volume énorme (de l'ordre du pétaoctet), fédérant souvent plusieurs bases de données réparties à différentes localisations. Ces entrepôts de données accumulent continûment les données provenant de systèmes transactionnels et les préparent pour l'analyse. Ils contiennent ainsi des données non volatiles, et constituent un archivage utile à la prise de décision.

Afin d'aider les décideurs à naviguer dans ces gros volumes de données, des outils pourvus d'une interface intuitive, avec plusieurs modes de visualisation ont été créés. C'est le cas, notamment, des outils OLAP (On-Line Analytical Processing). OLAP est un terme inventé par Codd *et al.* [1993] et ensuite repris par Pendse [2005b] pour

désigner une approche d'analyse rapide de données multidimensionnelles. Un système décisionnel comprend différents composants, allant des sources transactionnelles aux outils d'analyse avec l'entrepôt de données au centre (cf. Fig. 1.1).

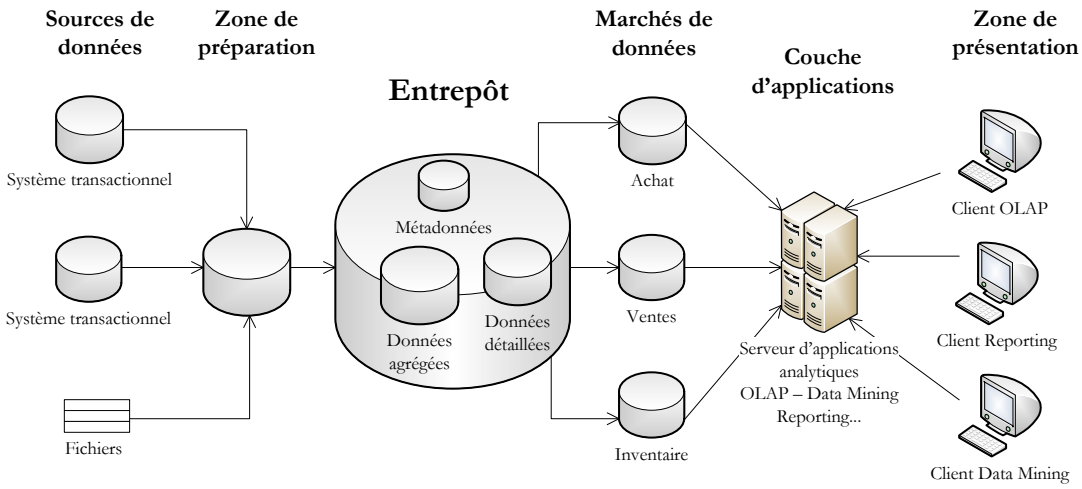


FIG. 1.1 – Exemple d'architecture d'un système décisionnel, inspiré de [Oracle, 2002]

Les systèmes OLAP ne géraient au départ que des données textuelles et numériques, et commencent aujourd'hui à intégrer les données spatiales. Il a en effet été constaté qu'environ 80% de l'information stockée dans les entrepôts de données possède une référence spatiale [Franklin, 1992]. L'ajout de la composante spatiale est un atout supplémentaire pour les décideurs, permettant la visualisation, l'analyse spatiale, etc. Aussi une nouvelle génération d'outils OLAP intégrant les données spatiales est née : les outils SOLAP (Spatial On-Line Analitical Processing). Ils proviennent du couplage entre les fonctionnalités d'affichage et d'analyse des technologies SIG (Système d'Information Géographique) avec celles du OLAP [Rivest *et al.*, 2005a]. Les entrepôts de données destinés à être exploités par des outils SOLAP contiennent donc également des données spatiales. Les récents progrès dans le stockage des données spatiales au sein des systèmes de gestion de bases de données (PostGIS, Oracle Spatial, etc.) permettent de stocker directement les données spatiales dans l'entrepôt.

Aujourd'hui, le monde décisionnel se doit de répondre à deux attentes majeures de la part des utilisateurs : l'exigence d'un temps de réponse rapide et de réponses les

plus à jour possible, voire en temps réel [Kennedy, 2003]. La gestion du temps réel est une préoccupation récente dans le monde des entrepôts de données et ouvre la porte à de nombreuses applications.

Le besoin en temps de réponse rapide est en grande partie comblé, car la structure de l'entrepôt de données a été conçue pour faciliter les analyses et accélérer le temps de réponse. L'entrepôt est désormais organisé sous forme de cube (ou plus précisément d'hypercube), représentation abstraite des données, dans laquelle chaque axe correspond à une dimension organisée en hiérarchies et chaque cellule du cube à un fait, rassemblant un membre de chaque dimension (donc un membre par axe) et une ou plusieurs mesures. L'ensemble des mesures est stocké dans une table de faits. Elles peuvent être agrégées selon les hiérarchies des dimensions et regroupées en mesures agrégées, dans des tables de faits agrégés ou au sein d'une même table de faits. On parle alors d'entrepôts de données multidimensionnels, car leur structure est dénormalisée (contrairement aux systèmes transactionnels classiques) pour augmenter la redondance et réduire le nombre de jointures coûteuses en temps de calcul pour certaines requêtes. Afin d'accélérer davantage la rapidité des requêtes, les mesures agrégées sont souvent précalculées et stockées dans le cube, ce qui a pour inconvénient d'accroître considérablement sa taille [Shukla *et al.*, 1996].

Les systèmes transactionnels recueillent continuellement des nouvelles données, ou des mises à jour qui doivent être propagées vers l'entrepôt de données. La mise à jour des entrepôts de données non spatiaux fait l'objet de nombreuses recherches [Mumick *et al.*, 1997; Vaisman, 2001] depuis plusieurs années. Le processus est complexe et nécessite beaucoup de temps de calcul (quelques heures), ceci en grande partie à cause de la structure dénormalisée du cube. La solution la plus répandue consiste à reconstruire tout l'entrepôt chaque nuit [Adamson, 2006], mais l'entrepôt ne reflète plus la réalité durant la journée et le temps de calcul requis pour le processus devient trop important au fur et à mesure que l'entrepôt grossit [Shukla *et al.*, 1996; Pendse, 2005a]. Parmi les solutions alternatives, celle qui paraît la plus prometteuse est la technique de mise à jour incrémentielle qui consiste à ne propager que les mises à jour et non à reconstruire le cube à partir de toutes les données [Thierney, 2006]. Le terme

incrémentiel signifie en effet : « qui fonctionne par ajout d'éléments, c.-à-d. quantité constante ajoutée à la valeur d'une variable à chaque exécution d'une instruction »[1]. Les techniques incrémentielles sont très utilisées dans un contexte d'entrepôts de données *temps réel* dans lesquels le flux de données entre les données source et l'entrepôt est quasi continu [Kimball et Caserta, 2004].

Si aujourd'hui Internet est une gigantesque autoroute sur laquelle circulent des données principalement entre clients et serveurs, on y voit se construire et grossir une autre voie de communication destinée aux échanges entre les systèmes, mais l'effort à fournir pour faire communiquer des systèmes hétérogènes est important. C'est dans cette optique que sont récemment apparues les architectures orientées service (SOA pour Service Oriented Architecture), dans lesquelles une application est constituée de services qui s'échangent des messages entre elles. La spécification des services web énoncée par le World Wide Web Consortium (W3C) s'inscrit dans cette architecture SOA, les services web permettent l'interopérabilité entre différents logiciels fonctionnant sur différentes plateformes. D'autres standards énoncés par l'OGC (Open Geospatial Consortium) ont vu le jour pour les données spatiales et leurs traitements, ils sont de plus en plus utilisés dans l'industrie pour la gestion et la diffusion des données spatiales.

La mise à disposition de processus de mise à jour incrémentiel des cubes de données spatiales sous forme de service web apporterait une brique de plus dans l'élaboration de l'architecture orientée service GeoSOA, au coeur des activités de recherche de certains chercheurs (Dr. Thierry Badard *et al.*). Cette architecture permet la gestion/diffusion de données géospatiales transactionnelles et décisionnelles et la conception d'applications géodécisionnelles dans un contexte de mobilité pour enrichir et mieux supporter la prise de décision.

[1] Larousse

1.2 La problématique de mise à jour

Le terme *mise à jour* est utilisé dans de nombreux contextes comme l'informatique, l'édition, le journalisme, la cartographie, etc. : pratiquement tous les domaines liés à l'information dans son ensemble. Sa signification varie dépendamment du contexte dans lequel il est employé. Aussi, une mise à jour de logiciel n'a pas exactement le même sens qu'une mise à jour de façade de bâtiment. La mise à jour de logiciel existe à elle seule, c'est un ajout, un correctif développé dans le but d'améliorer le logiciel alors que la mise à jour d'une façade représente plutôt le travail effectué sur la façade du bâtiment pour la nettoyer et lui redonner sa splendeur passée. Il existe donc des différences entre les emplois du terme mise à jour, mais on leur distingue un sens commun : celui d'améliorer l'élément et le « faire correspondre à l'état du savoir, du progrès, des derniers développements. »[2]

En **cartographie**, la mise à jour est définie comme la « rectification de certains éléments ou de certaines zones d'une carte, afin de tenir compte de modifications importantes, mais localisées, survenues dans la zone cartographiée »[3]. En **informatique**, une mise à jour désigne à la fois le correctif (fichier, programme) destiné à effectuer la mise à jour d'un logiciel et la nouvelle version à jour de ce logiciel. Cette nouvelle version comporte des modifications mineures par rapport à la version précédente, notamment dans le but de corriger certaines anomalies de fonctionnement[4]. Enfin, en **bases de données**, la mise à jour désigne le fait de faire un ajout, une modification ou une suppression d'enregistrements[4].

L'expression « mettre à jour » est fréquemment confondue avec des termes proches, mais dont la signification diffère légèrement :

* Réviser : examiner à nouveau pour apporter des modifications
* Modifier : apporter de légers changements
* Évoluer : subir des transformations, changer de caractère

[2] Dictionnaire Wiktionnary.

[3] Office Québécois de la Langue Française.

[4] Définitions provenant du dictionnaire électronique Antidote.

* Maintenir : entretenir qqch. et mettre en oeuvre des moyens pour entretenir qqch.
* Rafraîchir : redonner de la fraîcheur, de l'éclat à ce qui est défraîchi
* Corriger : rectifier les erreurs, améliorer
* Moderniser : rendre moderne[5]

Le meilleur synonyme pour « mettre à jour » serait « actualiser » qui signifie adapter à l'époque actuelle.

La notion de *mise à jour* est souvent évoquée dans la littérature concernant les entrepôts de données multidimensionnels non spatiaux, cependant les recherche se sont focalisées sur des parties du processus et aucune recherche n'aborde le processus global de mise à jour de l'entrepôt [Bouzeghoub *et al.*, 1999] : certains examinent les évolutions de schéma [Blaschka *et al.*, 1999; Bédard *et al.*, 2003; Bakillah, 2007; Favre *et al.*, 2007], d'autres se concentrent sur la reconstruction des agrégations [Mumick *et al.*, 1997; Rafanelli, 2003], sur l'évolution des données au sein des dimensions [Vaisman, 2001] ou encore sur l'évolution des stratégies d'affaires [Imhoff *et al.*, 2003]. Le sujet est plus globalement abordé sous l'expression « maintenance de l'entrepôt de données » qui est « l'action de garder les données à l'état actuel afin qu'elles représentent la vraie situation opérationnelle de la compagnie » (traduction libre de [Kimball et Caserta, 2004]). Bouzeghoub *et al.* [1999] déplorent la confusion régnant autour du sujet et le fait que le processus de rafraîchissement de l'entrepôt soit souvent réduit à un problème de maintenance des agrégats ou de peuplement de l'entrepôt. Il en résulte un manque d'opérateurs dédiés à la mise à jour (à l'instar du update SQL[6] pour les bases de données relationnelles).

Un entrepôt de données est entouré de plusieurs systèmes effectuant différentes tâches [Kimball et Ross, 2002]. Le système ETL (Extract-Transform-Load), situé dans la zone de préparation (Fig. 1.1) est la fondation même de l'entrepôt de données et consomme 70% des ressources nécessaires à sa création et sa maintenance ;

[5] Ces sept définitions proviennent du dictionnaire électronique Antidote.
[6] Structured Query Language

c'est un processus très exigeant en temps de calcul : il extrait les données des systèmes sources, les transforme afin qu'elles soient organisées de façon conforme au schéma de l'entrepôt de données et les charge dans l'entrepôt. C'est donc l'ETL qui va se charger de la mise à jour de l'entrepôt de données en réponse à des modifications sur les données sources. Si sa mission est simple à comprendre, ce processus peut devenir une tâche très complexe du fait des milliers de cas possibles à traiter, dépendant chacun des sources de données, des stratégies d'affaires, des logiciels, etc. [Kimball et Caserta, 2004]. Il est encore plus délicat dans le cas des données géospatiales, en raison de la complexité inhérente à ce type de données.

Face à cette complexité, une solution simple a été rapidement adoptée pour mettre à jour les données structurées en cube : reconstruire entièrement le cube durant la nuit [Kimball et Ross, 2002]. Cependant, cette technique est sur le point de tomber en désuétude pour plusieurs raisons :

- **Le temps de recalcul du cube est proportionnel à la taille du cube**, qui ne cesse de croître, car un entrepôt de données contient des données non volatiles, il les accumule afin de garder un historique. Au fil du temps, le cube grossit et la reconstruction du cube prend de plus en plus de temps.
- **Le besoin de réponses reflétant le plus possible la réalité**. Les données sont saisies à une fréquence de plus en plus élevée dans les systèmes transactionnels et doivent être propagées de plus en plus rapidement dans l'entrepôt. Avec cette technique, le cube ne reflète plus la réalité durant le jour, or les besoins d'analyse intrajournaliers sont de plus en plus fréquents (p. ex. pour analyser la fréquentation des transports en commun en fonction des heures).

Si de nombreux travaux de recherche se sont intéressés à optimiser le processus de mise à jour dans le cas des cubes de données non spatiales (cf. chapitre 3), le domaine des entrepôts de données spatiales est encore très jeune (les études ont commencé avec Caron [1998]). Aujourd'hui la conception des cubes de données spatiales a atteint une certaine maturité avec les notions introduites par Bédard *et al.* [2001] mais **le thème de la mise à jour a été peu abordé** et aucune méthode ni classification n'ont été proposées [Bédard *et al.*, 2003]. Lambert [2006] propose de former un cube jour-

nalier à intégrer ensuite dans le cube historique et Rageul [2007] étudie les mises à jour de structure de l'entrepôt au cours de l'analyse. Ces recherches ont permis d'aborder certains problèmes, mais n'excluent pas au final la reconstruction du cube, et seul le cas des ajouts de données y a jusqu'à présent été traité. De plus en plus d'applications imposent d'optimiser le processus de mise à jour dans les bases de données géodécisionnelles [Bédard *et al.*, 2001; Lambert, 2006]. De plus, si aujourd'hui on traite peu des mesures spatiales dans les cubes de données SOLAP, où le rôle joué par les données spatiales géométriques est essentiellement destiné à la représentation spatiale [Rivest *et al.*, 2001], celles-ci sont assurément le futur des cubes SOLAP, qui, en exploitant tout le potentiel des données spatiales, fourniraient des possibilités d'analyse spatiale décisionnelle inédites.

Proposer une typologie clarifiant les termes autour de la mise à jour des cubes de données permettrait d'établir de bonnes bases pour cerner les problèmes entravant la mise à jour des cubes de données spatiales. La proposition de méthodes et de processus de mise à jour incrémentielle dans le cas des cubes de données spatiales pourrait être un défi intéressant et pertinent. Il semblerait alors judicieux et raisonnable d'étudier et de se baser sur les méthodes incrémentielles établies pour les cubes non spatiaux afin de les adapter aux cubes spatiaux, et, le cas échéant, de suggérer de nouvelles techniques. La méthode incrémentielle permettrait d'envisager de plus gros cubes de données avec des mesures spatiales complexes et de se tourner vers des applications temps réel.

Notre problématique de recherche concerne donc **la clarification du terme de « mise à jour de cube » et l'optimisation des méthodes actuelles de « mise à jour de cubes spatiaux »**.

1.3 Objectifs

La problématique de recherche exposée en 1.2 fait état d'un manque de méthodes efficaces pour mettre à jour les cubes spatiaux. L'objectif principal de notre recherche est donc de **proposer des méthodes pour optimiser la mise à jour des cubes spatiaux** suite aux changements intervenus sur les données transactionnelles provenant de diverses sources et alimentant le cube. Le processus actuel n'est pas optimisé pour les cubes spatiaux, puisque la mise à jour s'effectue par une reconstruction complète du cube et prend donc énormément de temps, même s'il s'agit d'une mise à jour minime. Notre but est de nous concentrer sur **des techniques incrémentielles** pour ne propager que les effets des modifications sur les données sources dans le cube, afin de réduire le nombre de calculs, de transactions et donc le temps requis pour la mise à jour. Nous souhaitons également étudier la possibilité d'introduire le processus dans une **architecture orientée service** afin de le mettre disponible à distance et de façon interopérable sous forme de service web.

Cet objectif principal passe par l'accomplissement des objectifs intermédiaires suivants qui nous ont permis d'établir une méthodologie pour mener à bien la recherche :

* **Clarifier les expressions employées dans la littérature et cadrer notre sujet de recherche**. Nous élaborerons pour cela une typologie autour de l'expression *mise à jour de cube*.
* **Proposer une approche pour mettre à jour les cubes spatiaux de manière incrémentielle.**
* **Rendre le processus de mise à jour de cube interopérable et accessible à distance.** Pour y parvenir, nous concevrons un service web de mise à jour de cube.

1.4 Méthodologie

La méthode de recherche employée, exposée en Figure 1.2, s'inspire grandement des méthodes Agile d'ingénierie logicielle puisque le projet de recherche implique plusieurs développements informatiques et comporte des cycles itératifs [Larman, 2004] permettant de revenir sur la conception de nos processus.
Dans un premier temps, une revue de littérature nous a permis de mieux assimiler les concepts relatifs aux entrepôts de données décisionnels, et de constituer un premier état de l'art des solutions existantes pour mettre à jour des entrepôts de données non spatiales, puis un second état de l'art concernant la mise à jour des entrepôts de données spatiales. La revue de littérature s'est également arrêtée sur les concepts liés à la mise à jour pour établir une typologie de la *mise à jour de cubes de données*, puis sur les concepts propres aux cubes de données spatiales. Après cette étape, vient le coeur de la recherche : la proposition de méthodes et processus de mises à jour en concordance avec la typologie énoncée. Cette étape s'accompagne de la constitution des jeux de données et des choix techniques pour l'implantation du prototype. Ensuite, la conception de données sources et de cubes de données a été réalisée pour les tests et les processus de mise à jour définis dans l'étape précédente implantés dans un outil ETL afin de procéder à des expérimentations en vue de valider nos processus. Les expérimentations menées à l'aide des divers jeux de données nous ont permis de redéfinir de manière itérative nos méthodes et processus de mise à jour. Enfin, après avoir obtenu des résultats satisfaisants lors des tests précédents, les processus de mise à jour ont été déployés sous forme de service web pour être accessibles à distance et de façon interopérable. La validation de cette dernière étape complète notre preuve de concept.

Cette recherche a été réalisée dans le cadre d'une subvention offerte pour le projet « Un outil web interactif pour mieux comprendre les impacts des changements climatiques sur la santé publique » qui regroupe quatre chercheurs principaux : le Dr. Pierre Gosselin (leader du projet) de l'INSPQ (Institut de Santé Publique du Québec) et les Dr. Thierry Badard (coleader), Yvan Bédard et Jacynthe Pouliot du Département

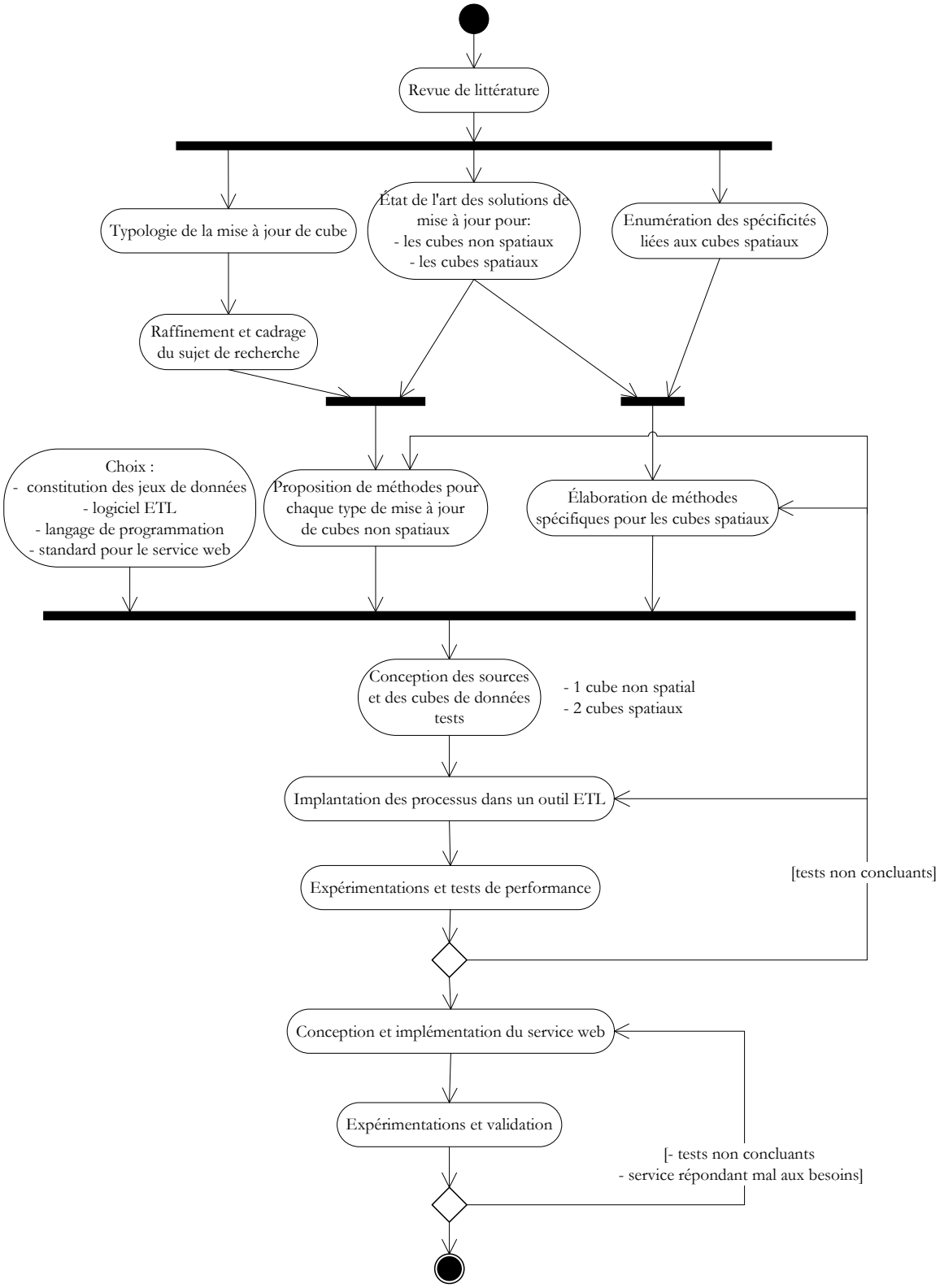


FIG. 1.2 – Diagramme d'activités UML représentant la méthodologie suivie

de Sciences géomatiques de l'Université Laval. L'outil web, conçu notamment à l'aide d'un outil SOLAP, a pour fonctionnalités l'analyse et la consultation des données géolocalisées de nature sociosanitaire et environnementale afin de mieux comprendre les interactions entre la santé et les changements climatiques. Les données relatives au climat changent constamment, par exemple la température ou l'humidité de l'air varient continuellement au cours d'une même journée. Pour capter ces variations, des relevés doivent être effectués sur des intervalles de temps très petits et à une fréquence élevée. La recherche menée au cours de cette maîtrise a pour but de fournir des données les plus à jour possible aux décideurs de santé publique. Le présent projet de maîtrise s'inscrit tout d'abord dans les recherches du groupe GeoSOA sur la gestion/diffusion de données géospatiales transactionnelles et décisionnelles et la conception d'applications géodécisionnelles dans un contexte de mobilité pour enrichir et mieux supporter la prise de décision. Il est également en lien avec les recherches sur l'extension des fonctionnalités du SOLAP menées au sein de la chaire industrielle CRNSG en bases de données décisionnelles dirigée par le Dr. Yvan Bédard.

1.5 Description des chapitres du mémoire

Le présent mémoire est structuré en cinq chapitres.

Le chapitre 2 fait figure d'entrée en matière, il définit la notion de mise à jour dans un contexte transactionnel, puis s'attarde sur le concept dans un contexte décisionnel. Les définitions entourant le monde décisionnel sont fournies et une typologie de la *mise à jour de cube* est formulée, ce qui permettra de cadrer le projet de recherche. Les problèmes entravant la mise à jour de cube spatiaux et non spatiaux seront exposés.

Le chapitre 3 présente de façon détaillée les méthodes existantes pour la mise à jour de cube dans les bases de données décisionnelles non spatiales et les techniques employées seront détaillées. Le chapitre expliquera la nécessité de proposer des solutions pour la mise à jour des cubes spatiaux.

Le chapitre 4 explicite la modélisation conceptuelle de notre solution, soit les méthodes et processus proposés pour la mise à jour incrémentielle de cube spatial.

La connaissance des problèmes liés aux cubes spatiaux nous permettra de proposer des solutions adaptées pour leur mise à jour. Enfin, les méthodes et procédés pour intégrer la fonctionnalité de mise à jour de cube dans une architecture orientée services seront explicités.

Le chapitre 5 retrace l'implantation de notre solution, à commencer par les choix technologiques effectués en terme de jeux de données, de système de gestion de base de données, d'outil ETL, de langage de programmation et de spécification pour le service web et les raisons de ces choix. Les différents moyens employés pour faciliter le processus seront explicités. Ensuite, une partie décrira l'implantation des processus de mise à jour dans l'outil ETL et une autre l'élaboration du service web de mise à jour, de son implémentation à son déploiement. Pour finir, les résultats des tests de performance seront fournis et commentés.

Enfin, le chapitre 6 conclut ce mémoire par une synthèse des réflexions et études menées dans le cadre de ce travail de recherche, expose et discute nos conclusions sur les solutions proposées et apporte un regard critique sur le travail accompli et les méthodes employées. Il fournit un certain nombre de perspectives concernant les futurs travaux de recherche dans le domaine des bases de données géodécisionnelles et leurs nouvelles applications émergentes, vers le temps réel et la mobilité.

Chapitre 2

La notion de mise à jour

Ce chapitre pose les bases de notre recherche en explicitant la notion de mise à jour. Nous étudierons la procédure dans les bases de données transactionnelles et dans les bases de données décisionnelles afin de mieux prendre conscience des différences significatives touchant le déroulement et les spécificités de la mise à jour dans ces deux contextes (sections 2.1 et 2.2). Nous donnerons une définition de l'expression *mise à jour de cube* et une typologie assez complète des modifications possibles (sous-section 2.2.3), cette typologie constituera l'assise de notre recherche et lèvera les ambiguïtés persistant autour du sujet. Nous détaillerons ensuite les étapes du processus de mise à jour de cube (section 2.2.4) et exposerons un état de l'art des différentes stratégies de mise à jour dans le domaine des entrepôts de données non spatiales (section 2.2.5). Pour finir, les problèmes entravant la mise à jour des cubes de données non spatiales et des cubes de données spatiales seront exposés (section 2.3).

2.1 La mise à jour dans un contexte transactionnel

Les systèmes transactionnels subissent souvent des mises à jour à une fréquence élevée, en particulier dans le cas de systèmes temps-réel. On pourrait penser ici aux transactions bancaires, aux statistiques de fréquentation des sites web, etc. Ils reposent pour la plupart sur des bases de données relationnelles et fonctionnent par enregistre-

ments dans des tables relationnelles. La mise à jour de données y est vue comme un ajout, une modification ou une suppression de données. Les opérations sur les données sont régies par des ordres exprimés dans le langage SQL. Ces ordres sont au nombre de trois : INSERT pour l'insertion de nouveaux enregistrements (nouvelles lignes) dans une table, DELETE pour la suppression et enfin UPDATE qui permet de modifier tout ou une partie des valeurs des colonnes. D'ordinaire, les systèmes transactionnels ne stockent pas d'historique.

Il est possible de conserver un historique des mises à jour en intégrant des notions de temps. **Les bases de données temporelles** sont des bases de données comprenant un modèle de données pour le temps. Les aspects temporels intégrés reposent essentiellement sur les notions de « temps valide » et de « date de transaction » [Dyreson *et al.*, 1994]. Le « temps valide » dénote la période de temps durant laquelle un enregistrement est valide en correspondance avec le monde réel ou des spécifications. Cette période est souvent implantée sous forme des deux attributs « date de naissance » et « date de mort ». La « date de transaction » est la date exacte à laquelle la transaction a eu lieu, c.-à-d. la date de stockage de l'enregistrement dans la base de données. Intégrer la référence temporelle rend le processus de mise à jour plus complexe en ce sens qu'il faut maintenir un historique. Pour cela, on opte souvent pour des techniques de versionnement dont voici les trois principales [Cellary et Jomier, 1990; Eder et Koncilia, 2003; Pouliot *et al.*, 2004] :

- par classe d'objets, introduit énormément de redondance.
- par occurrence, introduit également de la redondance, mais moins.
- par attributs (une table spéciale est réservée aux modifications subies par l'attribut et la valeur de l'attribut est remplacée dans la table principale). Cette dernière est plus difficile à mettre en oeuvre.

S'il est assez facile de mettre à jour des données non spatiales, le processus de mise à jour des **bases de données à référence spatiale** est très complexe et fait toujours l'objet de recherches [Badard, 2000; Worboys, 1998; Tøssebro et Güting; Pouliot *et al.*, 2004; Jeansoulin *et al.*, 2000]. Les bases de données géographiques ou bases de données

à référence spatiale sont conçues pour gérer les objets spatiaux, elles contiennent des modélisations et des structurations particulières dans lesquelles la géométrie occupe une place primordiale. Le processus de mise à jour de données spatiales met en jeu des techniques propres aux géométries comme l'appariement de points, de lignes et de polygones et doit prendre en compte la nature des modifications dans le monde réel [Badard, 2000]. Badard [2000] et Pouliot *et al.* [2004] ont établi des classifications des modifications sur les données spatiales et proposent des méthodes incrémentielles de propagation des mises à jour. Cette propagation fait, entre autres, appel à des techniques d'intégration de données. L'intégration de données spatiales est une problématique à elle seule. Les logiciels capables de manipuler beaucoup de formats de données spatiales sont peu nombreux : *FME* de la société Safe Software, l'extension *ArcGIS Data Interoperability Extension* d'ESRI ou le logiciel OpenSource *Spatial Data Integrator* proposé par CamptoCamp et Talend.

2.2 La mise à jour dans un contexte décisionnel

La mise à jour est un des processus les plus importants dans la gestion des entrepôts de données : « Most of the design decisions are then concerned by the choice of data structures and update techniques that optimise the refreshment of the data warehouse. » [Bouzeghoub *et al.*, 1999]. Les entrepôts de données ont deux particularités principales qui complexifient leur mise à jour : la dénormalisation des données et la préservation de l'historique. Dans un souci de clarté, il convient de préciser que les termes *mise à jour de cube* et *mise à jour d'entrepôt de données* n'ont pas la même signification : la *mise à jour de l'entrepôt de données* (également désignée par l'expression « rafraîchissement de l'entrepôt de données ») est la mise à jour du système global, tandis que la *mise à jour de cube*, sujet qui nous intéresse dans cette recherche, désigne le processus de mise à jour des données constituant le cube, ces deux processus sont donc très différents l'un de l'autre. Ces concepts seront détaillés en 2.2.3.

2.2.1 Définitions entourant les cubes de données

Afin de bien comprendre la notion de *mise à jour des cubes de données*, il convient de rappeler certaines définitions essentielles.

Entrepôt de données :

Conceptuellement, un entrepôt de données est une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour le support d'un processus d'aide à la décision [Inmon, 2005]. Cependant, dans la pratique, deux paradigmes existent : celui de Inmon et celui de Kimball. Pour Inmon [2005], une entreprise possède un unique entrepôt de données organisé selon la troisième forme normale et les marchés de données sont constitués à partir de cet entrepôt unique, tandis que pour Kimball et Ross [2002], l'entrepôt de données est l'ensemble des marchés de données de l'entreprise et, de plus, l'information y est stockée selon un modèle multidimensionnel. La définition de Kimball est la plus adoptée aujourd'hui, elle existe sous différentes formes, mais la mission de l'entrepôt reste toujours la même : entreposer les données afin de mieux supporter la prise de décision. Nous adoptons également la définition proposée par Kimball et Ross [2002]. En plus de l'entrepôt, l'environnement d'un entrepôt de données peut comprendre une aire de préparation des données (ETL), une aire de présentation des données et des outils d'accès aux données [Oracle, 2002], comme l'illustre la Figure1.1.

Cube de données :

Un cube (ou hypercube) de données est une structure de données permettant l'analyse rapide des données. Dans les cubes de données, les données sont organisées de manière multidimensionnelle : un cube est constitué de faits, appelés mesures groupées suivant des dimensions [Kimball et Ross, 2002]. Les faits sont stockés dans une table de faits et les dimensions dans des tables de dimensions. Les mesures peuvent être agrégées suivant les différents niveaux hiérarchiques des dimensions. Par exemple, un cube représentant la population du Canada par divisions administratives et âge peut également donner une représentation de la population par provinces et par groupes d'âge.

Un entrepôt de données peut être organisé sous forme de cube ou non. Dans le cas de notre recherche, l'entrepôt est organisé sous forme de cubes de données. On confond

facilement à tort le cube de données avec l'entrepôt de données, car parfois l'entrepôt de données ne contient qu'un seul cube[1].

Dans un système décisionnel, il existe plusieurs types de cubes de données : l'entrepôt de données et les marchés de données peuvent être organisés sous forme de cubes, stockés dans des SGBDs relationnels et les serveurs OLAP peuvent également contenir des cubes de données, souvent stockés dans un format propriétaire (cf. Fig. 2.1).

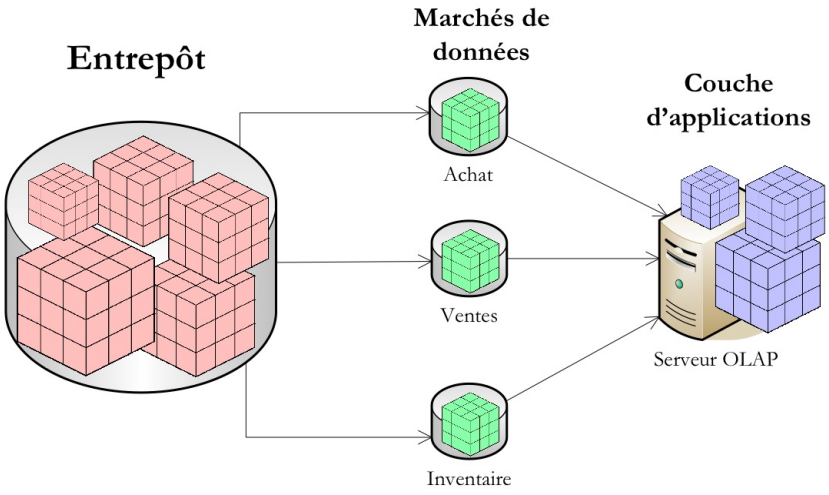


FIG. 2.1 – Les cubes de données d'un système décisionnel.

Schéma étoile :

Également connu sous le nom de « star-join schema », le schéma étoile est le modèle d'implantation multidimensionnel des données au sein du cube dans une base de données relationnelle le plus répandu. Chaque dimension est liée à une ou plusieurs tables de faits à l'aide de clés [Kimball et Ross, 2002]. Dans l'exemple donné en Figure 2.2, l'hypercube est composé de cinq axes ou dimensions : *age*, *annee*, *sexe*, *statut* et *province* et possède une table de faits : *Fait_population*. Les mesures *naissances*, *morts* et *population* sont stockées dans la table de faits.

Schéma en flocon :

Également connu sous le nom de « snowflake schema », le schéma en flocon constitue une

[1]D'où une confusion entre les termes *mise à jour de cube* et *mise à jour d'entrepôt de données*

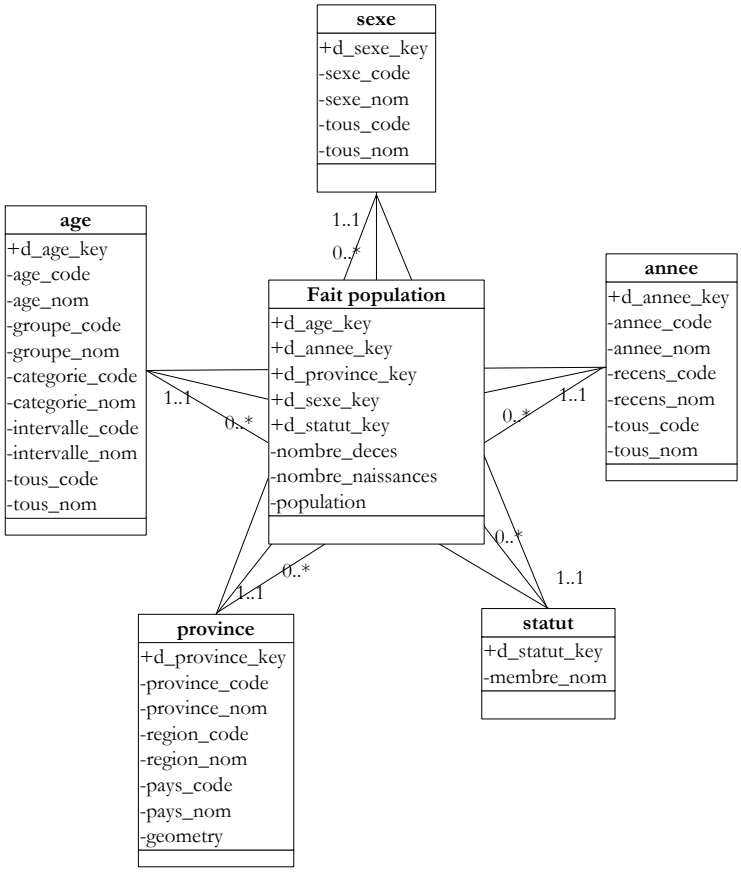


FIG. 2.2 – Modèle multidimensionnel en étoile représentant la population d'un pays.

alternative au schéma étoile pour stocker les données d'une manière multidimensionnelle du cube et dans une base de données relationnelle. Le schéma flocon est tiré du schéma étoile, chaque dimension est liée à une ou plusieurs tables de faits à l'aide de clés, mais les dimensions sont décomposées en plusieurs tables, chaque table correspond à un niveau de la hiérarchie de dimension. Dans l'exemple donné en Figure 2.3, la dimension *province* contient une hiérarchie avec trois niveaux : *province*, *région* et *pays*.

Cube de données spatiales :

Les cubes de données spatiales sont des cubes de données où des membres de dimension ou bien des faits sont référencés spatialement et peuvent être représentés sur une carte [Bédard *et al.*, 2008]. Le cube de la Figure 2.2 est un cube de données spatiales, car il possède une dimension spatiale : *provinces*, elle contient un champ *geometry*. Cette dimension pourrait être illustrée sur une carte (cf. Annexe E, Figure E.1).

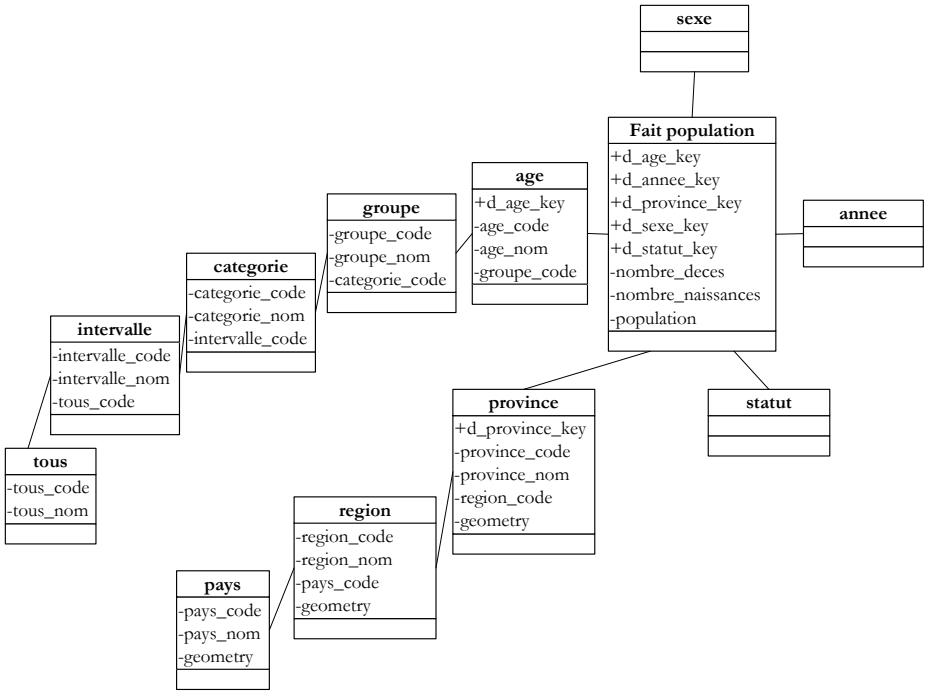


FIG. 2.3 – Modèle multidimensionnel en flocon représentant la population d'un pays.

Dans la suite de ce mémoire le terme cube désigne un cube de données situé dans l'entrepôt de données et stocké dans une base de données relationnelle. Le schéma en flocons est utilisé pour implanter des dimensions à hiérarchies ayant des relations complexes entre ses niveaux. Dans les autres cas, le schéma étoile est la forme d'implantation la plus utilisée en raison de sa facilité de compréhension et des meilleures performances. Nous essaierons de construire nos cubes sous forme de schéma étoile et le cas échéant, nous adopterons le schéma en flocons.

Cube de données pour la suite du mémoire :

1. Ensemble de tables organisées de manière multidimensionnelle.
2. Situé dans une base de données relationnelle en schéma étoile ou flocon.

2.2.2 Cadre étendu du sujet de recherche : la gestion de l'entrepôt de données

La gestion de l'entrepôt de données est une tâche plus complexe que sa construction, qui devrait être explicitée en détail dans la littérature ; or on déplore un manque d'ouvrages traitant le sujet dans sa totalité et une certaine confusion [Bouzeghoub *et al.*, 1999]. Elle est souvent reléguée à la fin de la planification de construction du système, avec les tests de performance et la livraison. Aucune méthode n'est fournie pour aider les personnes en charge de la gestion, une fois le système installé.

Nous exposons d'abord sous forme de schéma (Fig. 2.4) une synthèse des tâches composant la gestion des entrepôts de données, effectuée à l'aide des ouvrages suivants [Humphries *et al.*, 1998; Bouzeghoub *et al.*, 1999; Ponniah, 2001; Rafanelli, 2003; Scalzo, 2003; Imhoff *et al.*, 2003].

La gestion de l'entrepôt de données est divisée en deux parties : la **maintenance** et la **mise à niveau**.

La maintenance est l'ensemble des opérations qui sont jugées nécessaires pour garantir en tout temps le bon fonctionnement d'un système informatique, conformément à des spécifications définies[2]. Elle concerne donc les opérations courantes. La **maintenance d'un entrepôt de données** consiste à garantir le fonctionnement de tous les jours de l'entrepôt. On y inclut l'entretien du matériel, des logiciels (mises à jour, etc.), la gestion de l'espace de stockage, le support aux utilisateurs, la gestion du réseau, de la sécurité et des sauvegardes. La maintenance comprend également les opérations de vérification et d'adaptation des processus ETL et les opérations sur les cubes : la mise à jour des cubes et les corrections.
La mise à jour (plus connue dans le monde logiciel sous le terme *update*) est une « opération qui consiste à adapter le matériel ou le logiciel de manière à ce qu'il soit conforme aux développements les plus récents ou à modifier les données pour qu'elles reflètent l'information la plus actuelle. Le terme mise à jour désigne également

[2]Office Québécois de la Langue Française.

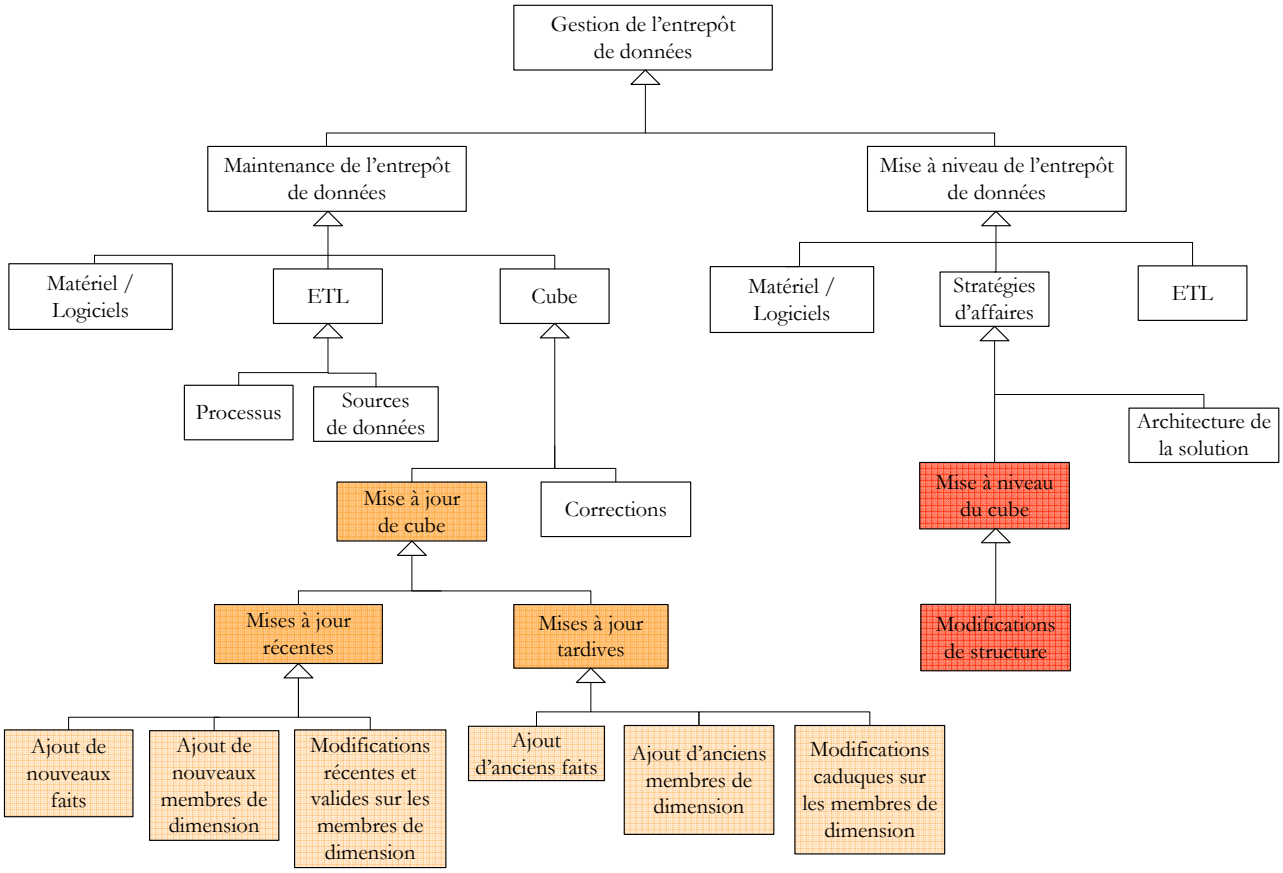


FIG. 2.4 – Gestion de l'entrepôt de données.

le résultat de l'opération. »[3] Contrairement à la mise à jour, la correction a lieu uniquement suite à des erreurs. Dans les entrepôts de données, les corrections sont rares, voire inexistantes. L'entrepôt a pour mission de garder l'historique des données ; or effectuer une correction efface cet historique. Nous pensons que les corrections sont possibles, car les données doivent être modifiables en cas d'erreurs majeures et éliminer totalement cette possibilité de corriger les erreurs reviendrait à garder une représentation biaisée de la réalité. Toutefois, nous ne nous intéressons pas aux corrections par la suite puisque nous estimons qu'elles sont occasionnelles marginales, comme l'affirment Kimball et Caserta [2004].

La mise à niveau (plus connue, sous le terme *upgrade*) est « l'opération qui consiste à remplacer un système informatique ou une partie de celui-ci par un modèle plus

[3] Office Québécois de la Langue Française, section informatique.

puissant ou par une version plus perfectionnée, ou encore à modifier ce système sur le plan matériel par l'ajout de nouveaux éléments [...] de manière qu'il soit plus [...] perfectionné. »[4]. La **mise à niveau d'un entrepôt de données** est une opération ponctuelle qui consiste à effectuer des modifications majeures sur l'entrepôt pour en créer une nouvelle version plus performante. Il s'agit d'une évolution de l'entrepôt de données. Elle comprend les évolutions apportées sur le matériel, les logiciels, le stockage, le réseau, les sauvegardes, la sécurité, le support technique et les modifications majeures apportées aux processus ETL. Un changement de stratégie d'affaires va également entraîner des modifications conséquentes sur l'entrepôt : il faudra soit revoir l'architecture globale de la solution soit mettre à niveau les cubes par une modification de leur structure ou bien les deux.

En résumé, l'expression *mise à jour* fait référence à des changements mineurs alors que la *mise à niveau* a lieu lorsque sont effectués des changements majeurs :

* Dans notre cas, on considère que les **changements mineurs** sur le cube sont des modifications qui n'auront pas d'impact sur les applications dérivées (ici : OLAP, SOLAP, Data Mining, etc.) et ne vont donc pas les rendre obsolètes. Ils sont transparents pour les serveurs OLAP, les clients, etc. : c.-à-d. toutes les applications dépendantes et se trouvant principalement dans la zone de présentation. L'usager du cube peut toujours profiter de toutes les fonctionnalités offertes par ces applications, elles pourront continuer à fonctionner sans avoir besoin d'être retestées et modifiées. Un changement mineur n'inclut pas non plus d'ajout de nouvelles fonctions puisqu'il faudrait alors revoir le soutien technique et la formation aux usagers.
* Les **changements majeurs** auront, quant à eux, un impact sur les applications dérivées, car vont invalider tout ou partie de l'application. Si l'on se place dans une application OLAP ou SOLAP, des modifications devront être apportées au serveur pour qu'il prenne en compte la nouvelle version du cube. Ce sont des changements qui impliquent une redéfinition du cube.

Pour illustrer nos propos, nous prenons l'exemple du serveur OLAP Mondrian fourni par la société Pentaho. Dans Mondrian, le cube est inclus dans un schéma défini par un fichier XML (cf. Annexe A) qui décrit, entre autres, toute la structure du cube (dimension, hiérarchies, niveaux, nom des tables relationnelles, nom des mesures, etc.). Toute modification touchant à la structure du cube aura des répercussions sur le fichier XML, il devra être modifié en conséquence, sinon le serveur Mondrian ne pourra plus accéder au cube. Il en est de même pour l'application Jmap SOLAP de la société Kheops Technologies ; des modifications devront être apportées dans l'administration de l'outil afin de prendre en compte les changements majeurs.

Au vu de ces explications, nous pouvons affirmer que les **changements mineurs** sont des modifications de **contenu** du cube (sur les données brutes) alors que les **changements majeurs** sont des modifications de **contenant** (structure du cube).

Kimball et Caserta [2004] donnent une définition voisine de l'expression *changements mineurs* : les *graceful modifications* (ou modifications élégantes). Ce sont des modifications qui n'affectent pas les requêtes des utilisateurs ou bien les applications dépendantes. Cependant, Kimball et Caserta [2004] qualifient l'augmentation de granularité des faits ou l'ajout d'une dimension de modifications élégantes, alors que ce sont des changements de structure, et donc pour nous, des changements majeurs.
Thierney [2006] introduit la notion de *mise à jour incrémentielle de cube* pour les cubes MOLAP situés dans le serveur MOLAP SAS. La fonctionnalité sera disponible dans la prochaine version (9.2) du logiciel SAS OLAP Server 9.2. Dans ce rapport technique, Thierney [2006] définit brièvement l'expression *cube update* (*mise à jour de cube*) et son contraire : la mise à jour ne doit pas contenir de changements majeurs de structure du cube et ne peut ni effacer, ni remplacer les données du cube. Suite à des changements majeurs de structure, le cube n'est plus le même et doit repasser par les étapes de développement et de test. La mise à jour autorise toutefois des changements

mineurs de structure [Thierney, 2006]. Il s'agit ici de la première tentative de définition de l'expression *cube update* (ou *mise à jour de cube*), les grandes lignes sont tracées, mais les règles sont imprécises (les expressions changements mineurs et majeurs ne sont pas définies) et doivent être revues ou éclaircies.

Afin de formuler une définition plus univoque de l'expression ***mise à jour de cube*** nous nous sommes inspirés de Thierney [2006] et des notions de *mise à jour* et *mise à niveau* de logiciel dans le contexte informatique. Si l'on ne peut comparer le cube avec un logiciel, ils possèdent des points communs : le cube et le logiciels ont des utilisateurs, des administrateurs, etc. Les explications apportées précédemment sur les changements mineurs et majeurs permettent de poser nos définitions pour les expressions *mise à jour de cube* et *mise à niveau de cube* :

> ***Mise à jour de cube* : évolution des données du cube**
>
> 1. contient uniquement des modifications sur les données (données de dimensions et faits).
> 2. fréquente, fait partie de l'utilisation usuelle du cube.
> 3. n'invalide pas les applications dépendantes et donc, comprend des *changements mineurs*.
> 4. est transparente pour l'usager.

Il existe deux types de mises à jour pour un cube, elles seront détaillées dans la typologie de *mise à jour de cube* en 2.2.3.2 :

1. les mises à jour récentes (fréquent, processus régulier).
2. les mises à jour tardives (rare).

La meilleure manière de clarifier l'expression *mise à jour de cube* est d'expliciter son opposé que l'on nomme : *mise à niveau de cube*.

***Mise à niveau de cube* : évolution du contenant**

1. contient des modifications du modèle d'implantation (nom des tables, format des données etc.) et/ou des hiérarchies de dimension (nom, organisation des hiérarchies etc.).
2. moins fréquente et plus ponctuelle : patches de cube et nouvelles versions.
3. invalide les applications dépendantes et donc, comprend des *changements majeurs*.
4. implique souvent une redéfinition du cube : changement de nature.

La mise à niveau du cube est une évolution du contenant du cube à des fins de corrections (patches) ou pour intégrer de nouvelles fonctionnalités (nouvelle version du cube). Une brève typologie sera fournie en 2.2.3.3.

Il existe cependant certains cas particuliers aux règles énoncées : si l'usager a prédéfini certaines requêtes dans son application, une mise à jour de cube peut invalider ces requêtes, il faut alors les modifier. Pour se prévaloir de toutes complications, des avertissements sur les modifications effectuées pourraient être envoyés aux utilisateurs [Levesque *et al.*, 2007]. Un exemple de requête invalidée est donné en Annexe B.

2.2.3 Définition de la mise à jour de cube

Nous avons situé le processus de mise à jour de cube dans le contexte plus global de mise à jour de l'entrepôt de données. Cette prise de recul était nécessaire à la compréhension et nous permet désormais de nous concentrer sur la mise à jour de cube afin d'établir une classification des types de mise à jour possibles. Nous aborderons brièvement la notion de temps, primordiale pour ensuite exposer notre typologie de

mise à jour de cube.

2.2.3.1 Notions temporelles

Le temps est un élément prépondérant dans le processus de mise à jour. Chaque étape de la mise à jour de cube est ancrée dans le temps. Dans un contexte de mise à jour de cube, il existe trois dates représentant des instants importants :

1. La date de modification du phénomène dans la réalité : $\mathbf{T_{chgt}}$. C'est la date à partir de laquelle le nouvel état de l'objet devient valide et l'ancien état prend fin. $\mathbf{T_{chgt}}$ correspond à la « date de naissance » présente dans la notion bitemporelle de « temps valide » vue au début du chapitre.
2. La date de saisie de la modification dans le système transactionnel (sources) : $\mathbf{T_{saisie}}$. $\mathbf{T_{saisie}}$ correspond à la « date de transaction » vue au début du chapitre.
3. La date de mise à jour du cube : $\mathbf{T_{maj}}$.

Les aspects temporels du processus de mise à jour sont schématisés en Figure 2.5. Entre $\mathbf{T_{chgt}}$ et la date exacte de fin de la mise à jour, le cube ne représente plus la réalité.

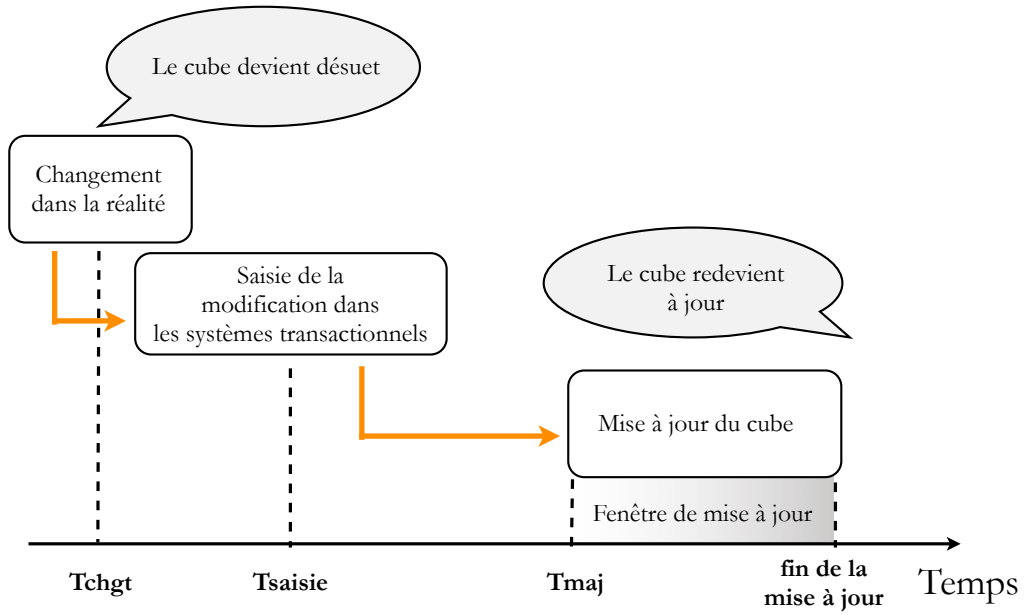


FIG. 2.5 – Temporalité dans le processus de mise à jour de cube.

2.2.3.2 Typologie de mise à jour de cube

Le cube va subir de nombreuses mises à jour au cours de son existence. Soit **n**, un entier ; la **n***ieme* mise à jour peut comprendre deux types de mises à jour (cf. 2.2.2) : des mises à jour récentes et des mises à jour tardives. On note $\mathbf{T_{chgt\ n}}$ la date d'une modification devant participer à la **n***ieme* mise à jour, $\mathbf{T_{saisie\ n}}$ la date de saisie dans le système transactionnel pour la **n***ieme* mise à jour de cube, $\mathbf{T_{maj\ n}}$ la date de la **n***ieme* mise à jour.

Intégration des mises à jour récentes

L'intégration des données récentes est un processus fréquent qui consiste à ajouter de nouvelles données dans le cube, correspondant à de nouvelles informations dans la réalité. Il s'agit du fonctionnement courant de la mise à jour, illustré en Figure 2.6 ; les rectangles foncés reliés par les flèches correspondent aux étapes de la **n***ieme* mise à jour.

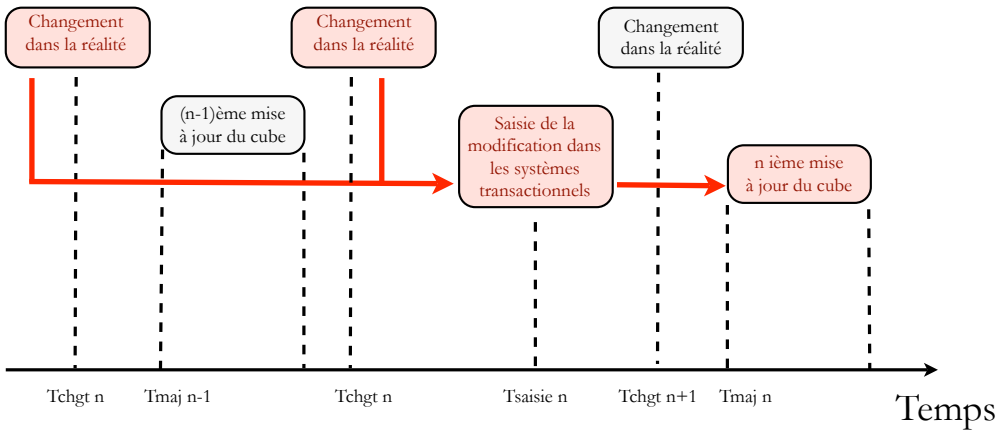


FIG. 2.6 – Intégration des mises à jour récentes dans le cube.

Les nouvelles données peuvent être des **nouveaux faits détaillés** ou des **modifications sur les membres de dimension**.

L'ajout de nouveaux faits détaillés est le type de mise à jour le plus fréquent et est souvent périodique. Les nouveaux faits sont souvent nombreux (plusieurs se dénombrent par milliers) et leur intégration implique l'ajout de nouveaux membres dans la dimen-

sion temporelle et le recalcul des agrégations. La mise à jour des faits concerne toujours les faits détaillés ; les faits agrégés ne sont pas modifiés directement, mais rafraîchis suite à l'ajout de nouveaux faits détaillés.

Les mises à jour sur les membres de dimension sont moins fréquentes et plus ponctuelles. Elles consistent soit à ajouter de nouveaux membres, soit à modifier des membres existants suite à une évolution de la réalité. Ces modifications n'ont pas d'effets sur les faits détaillés ou agrégés antérieurs[4] ; en effet les membres de dimension sont constitués de plusieurs occurrences possédant chacune une période de validité, et chaque fait concerne une seule occurrence.

Il existe six types de mises à jour sur les membres de dimension :

- ajout d'un membre
- modification de la valeur d'un attribut de membre
- suppression d'un membre
- fusion de **n** membres en **1** sur le même niveau[5]
- scission de **1** membre en **n** sur le même niveau[8]
- reclassification d'un membre dans la hiérarchie dimension[8]

Intégration des mises à jour tardives

Les mises à jour tardives sont des mises à jour qui auraient dû être intégrées dans le cube lors d'une mise à jour antérieure, mais qui pour des raisons inconnues n'ont pas pu être saisies dans le système transactionnel en temps voulu (p. ex. *pour un cube mis à jour toutes les nuits : la saisie au mois d'août dans le système transactionnel des ventes du 15 janvier*). Le processus est illustré en Figure 2.7 : les changements qui auraient dû être enregistrés lors de la $(\mathbf{n}-\mathbf{p})$*ieme* saisie pour participer à la $(\mathbf{n}-\mathbf{p})$*ieme* mise à jour du cube ont été saisis plus tard, lors de la $\mathbf{n}$*ieme* saisie et participeront donc à la $\mathbf{n}$*ieme* mise à jour. Ce type de mise à jour est rare.

Les mises à jour tardives peuvent être des **anciens faits détaillés** (cf. *Late arriving*

[4]Sauf si le client veut répercuter les mises à jour sur les faits antérieurs, mais cela constituerait plutôt une correction. Nous n'envisageons pas les corrections.

[5]Peut invalider les requêtes prédéfinies sur ce membre

fact rows [Kimball et Ross, 2002; Kimball et Caserta, 2004]) ou des **modifications caduques sur les membres de dimension** (cf. *Late arriving dimension rows* [Kimball et Ross, 2002; Kimball et Caserta, 2004]).

L'ajout d'anciens faits détaillés correspond à l'insertion de faits caduques dans la table de faits détaillés, à la suite de laquelle il est nécessaire de recalculer les agrégations.

Les modifications caduques sur les membres de dimension ne sont pas fréquentes, mais sont plus difficiles à intégrer dans le cube. S'il s'agit de modifier un membre existant, il faudra corriger les dates de validité des deux occurrences les plus proches dans le temps (antérieure et postérieure) du membre, corriger les clés attribuées dans la table de faits, et enfin, recalculer les agrégations. Dans le cas d'un membre non existant, la procédure est plus simple, car il suffit d'ajouter le membre dans la dimension.

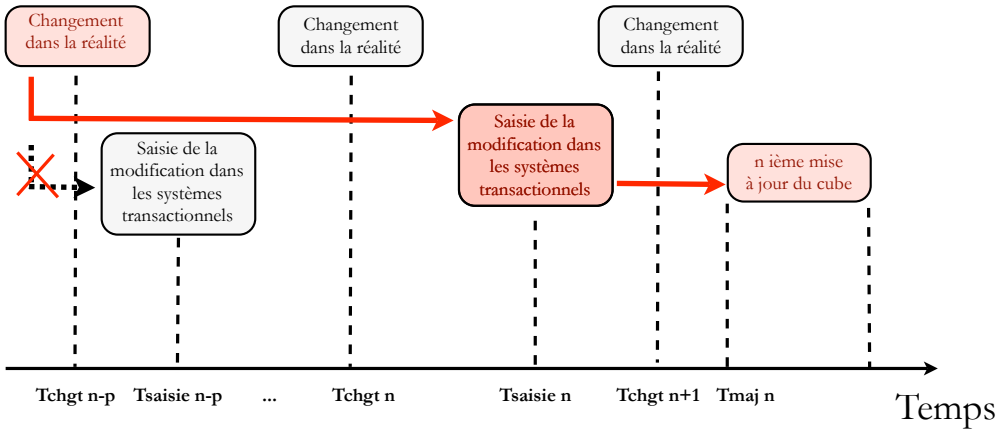


FIG. 2.7 – Intégration des mises à jour tardives dans le cube.

La typologie de *mise à jour de cube* énoncée est résumée dans le Tableau 2.1.

2.2.3.3 Brève typologie de mise à niveau de cube

De la même façon que l'on met à niveau un logiciel en le corrigeant ou en créant une nouvelle version incorporant de nouvelles fonctionnalités, on met à niveau un cube lorsque l'on corrige ses failles ou que l'on en crée une nouvelle version à partir de l'ancien, intégrant, par exemple, de nouvelles sources de données, de nouvelles mesures,

Types de mise à jour	**Mises à jour récentes**	**Mises à jour tardives**
ajout de faits	nouveaux faits	anciens faits
ajout de membres de dimension	nouveaux membres	anciens membres
modifications sur les membres de dimension	modifications récentes et valides	modifications caduques

TAB. 2.1 – Typologie de mise à jour de cube.

etc. : c.-à-d. des changements structuraux majeurs. La plupart des auteurs traitant les évolutions de schéma pour un cube [Blaschka *et al.*, 1999; Vaisman, 2001; Bédard *et al.*, 2003; Bakillah, 2007; Favre *et al.*, 2007] ont en fait abordé le processus de mise à niveau de cube. Nous donnons une liste sommaire des modifications touchant la structure du cube et pouvant invalider les applications dépendantes (cf. changements majeurs au 2.2.2) :

Modifications sur les dimensions

- ajouter / renommer / modifier le format / supprimer un attribut (propriété de membre)
- ajouter / renommer / supprimer une dimension
- ajouter / renommer / supprimer une hiérarchie
- ajouter / renommer / modifier le format / supprimer un niveau
- fusionner / scissionner / reclassifier des niveaux

Modifications sur la tables de faits détaillés

- ajouter / renommer / modifier le format / modifier la fonction / supprimer une mesure
- augmenter / diminuer la granularité
- renommer la table de faits

Modifications sur les faits agrégés

- ajouter / renommer / modifier / supprimer une table de faits agrégés

Parmi les modifications énoncées, certaines sont conséquentes (ex : augmentation de la granularité) et redéfinissent la nature du cube, elles font souvent suite à un chan-

gement de la stratégie d'affaires (ex : une redéfinition des besoins), et d'autres modifications auront moins d'impact (ex : l'ajout d'un attribut de membre). Cependant, tous les changements de structure sont susceptibles d'avoir des répercussions sur le fonctionnement du processus de mise à jour ; les cubes ayant subi des corrections et les nouvelles versions du cube devront repasser les tests de développement, les tests de déploiement et d'acceptation des utilisateurs. Il faudra avertir les usagers ou les former, afin qu'ils s'adaptent aux modifications.

2.2.4 Le processus global de mise à jour de cube

Bouzeghoub *et al.* [1999] expliquent que le processus est complexe et peut être modélisé comme un flux de travaux dont l'orchestration dépend fortement des produits disponibles pour l'ETL, de la stratégie d'affaires et du domaine d'application, mais également de la qualité requise en terme de fraîcheur des données et des différents besoins des utilisateurs.

Pour bien situer le contexte de la mise à jour de cube, nous en résumons les grandes étapes en Figure 2.8. Les principales étapes sont, dans l'ordre : l'extraction des mises à jour au niveau des données sources, la transformation des données de mise à jour afin qu'elles soient compatibles avec le schéma du cube, l'intégration de ces mises à jour dans le cube et la propagation de leurs effets au sein du cube.

La mise à jour du cube est gérée par trois systèmes : les systèmes sources, l'ETL et le SGBD relationnel cible contenant le cube. Les étapes principales internes au cube et leurs acteurs sont :

- l'extraction des mises à jour : sources ou ETL
- la transformation des données de mise à jour : ETL
- l'intégration des mises à jour dans le cube : ETL
- la propagation des effets des mises à jour au sein du cube : ETL et/ou SGBDR cible

Le superviseur de la mise à jour est l'outil ETL. Il s'occupe de nombreuses tâches externes (déclenchement, archivage, audit etc.) et internes (extraction,

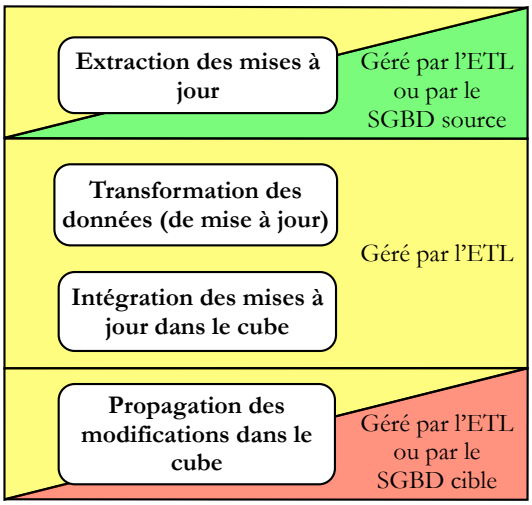


FIG. 2.8 – Étapes de la mise à jour de cube.

transformation et chargement des données, etc.) au cube concernant la mise à jour. Kimball et Caserta [2004] expliquent remarquablement bien le rôle de l'ETL par une analogie entre un restaurant et un entrepôt de données. L'ETL est la cuisine du restaurant (back room), elle fournit la nourriture (données) à la salle à manger (aire de présentation ou front room en anglais). La séparation entre la cuisine et la salle à manger est nette, il en est de même entre l'ETL (back room) et l'aire de présentation des données. Les deux sont physiquement, logiquement et administrativement séparées. L'accès aux données dans l'ETL est interdit aux utilisateurs, tout comme les clients n'ont pas le droit de pénétrer dans la cuisine du restaurant. L'ETL consomme 70% des ressources humaines et matérielles nécessaires à la création et à la maintenance de l'entrepôt. Nous aborderons uniquement les tâches d'extraction, d'intégration et de propagation définies ci-dessus. L'étape de transformation est utile lorsque les données sources et de l'entrepôt sont dans différents formats : par exemple les données sources dans des fichiers texte et les données de l'entrepôt dans un format de base de données. Toutes nos données sont stockées dans des bases de données, l'étape de transformation est inexistante.

Le processus de mise à jour du cube peut être vu comme une **propagation de données**, un flux de données entre les systèmes sources et les différentes

tables du cube. Cette propagation, illustrée en Figure 2.9 est à deux niveaux :

1. propagation des mises à jour des systèmes transactionnels vers le cube (modifications directes) : intégration des mises à jour dans le cube.
2. propagation des effets de l'étape précédente à l'intérieur du cube (modifications indirectes ou secondaires).

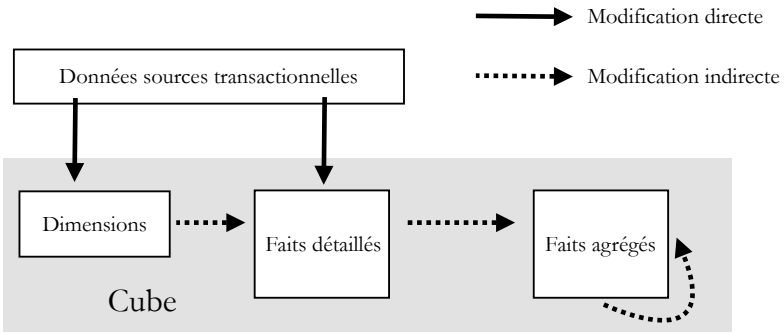


FIG. 2.9 – Processus de propagation des mises à jour dans un cube de données.

L'ordre d'exécution de ces étapes de propagation des données est fondamental et dépend de la nature de chaque mise à jour. Suite à la typologie énoncée en 2.2.3.2, l'ETL doit exécuter les mises à jour dans l'ordre suivant :

1. intégration des modifications tardives sur les dimension
2. intégration des vieux faits
3. intégration des modifications récentes sur les dimensions
4. intégration des nouveaux faits
5. rafraîchissement des agrégations

Bien que les étapes 1 et 2 nécessitent le recalcul des agrégations (5), celui-ci est fait à la fin. Il est préférable d'effectuer le rafraîchissement en une seule fois pour gagner du temps. La séquence d'exécution est illustrée en Figure 2.10.

Dès la mise à jour déclenchée, toutes les données (mesures et données de dimensions) deviennent inaccessibles. Les utilisateurs interrogent essentiellement les agrégations, on pourrait penser qu'ils doivent attendre la complétion de l'étape 5 pour y accéder. En fait, les utilisateur peuvent accéder aux données mises à jour dès la fin de l'étape 4

(saisie des nouveaux faits) et donc avant la fin de la mise à jour complète du cube. Ceci grâce au navigateur d'agrégation.

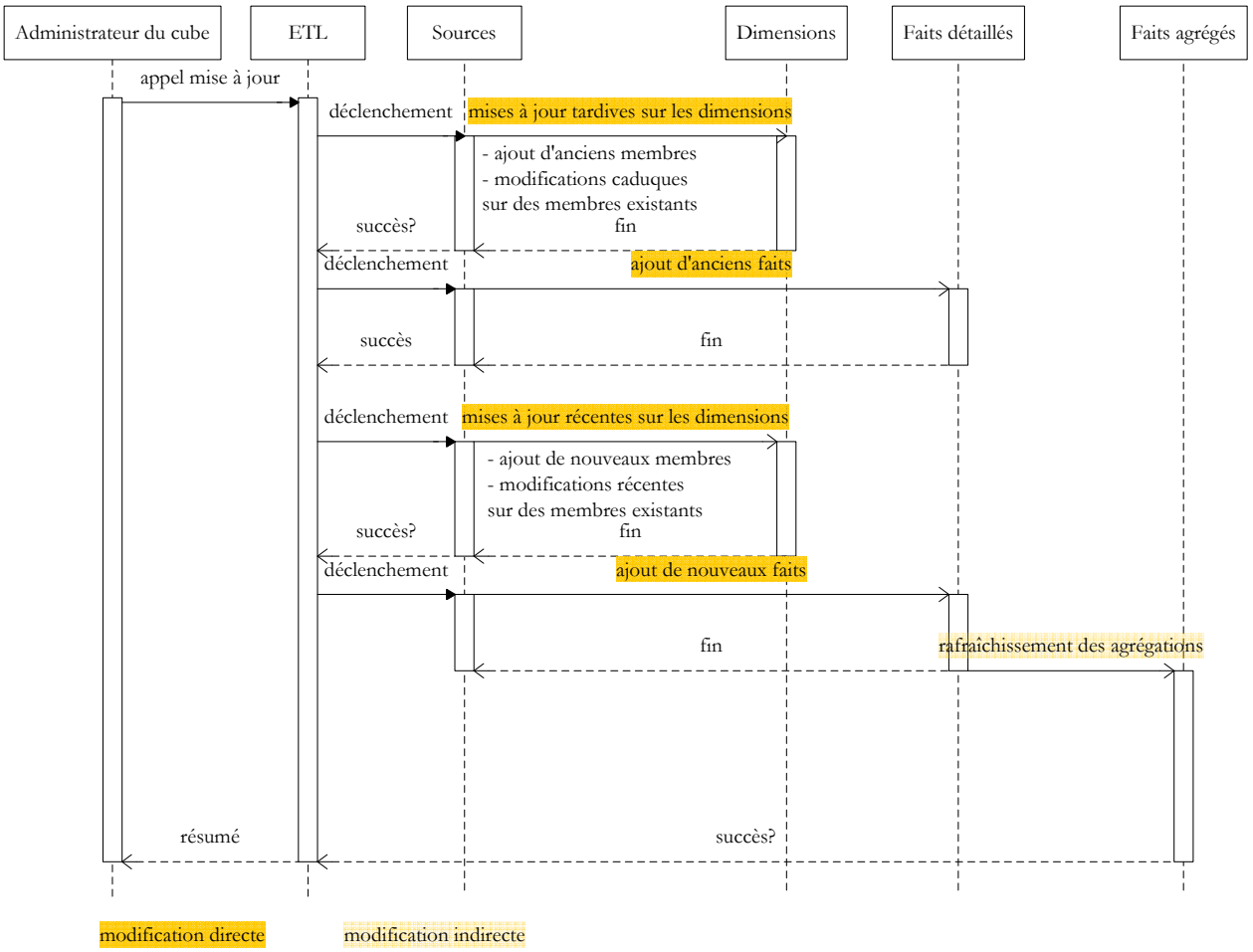


FIG. 2.10 – Séquence d'exécution de la mise à jour de cube.

En effet, les applications clientes ne requêtent pas directement sur les tables d'agrégation, les requêtes sont d'abord reçues et traitées par le navigateur d'agrégations (Fig. 2.11). Kimball [1995] définit le navigateur d'agrégations de la manière suivante :

> With an aggregate navigator, the end-user application now speaks "base-level" SQL and never attempts to call for an aggregate directly. Using metadata describing the data warehouse's portfolio of aggregates, the aggregate navigator transforms the base-level SQL into "aggregate-aware" SQL.

Les utilisateurs sont au courant de l'existence des agrégats au niveau conceptuel, mais pas de la manière dont ils sont stockés dans la base de données. Le navigateur évite les problèmes dus aux modifications de structure des agrégats et permet de mettre les agrégats hors ligne pendant leur rafraîchissement, puisque la table des faits détaillés est disponible et à jour.

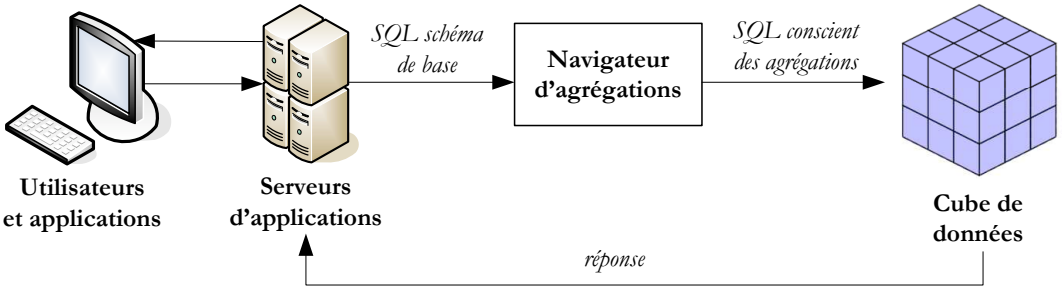


FIG. 2.11 – Le navigateur d'agrégations.
Inspiré de [Adamson, 2006].

Le navigateur d'agrégats peut être inclus dans l'application cliente (on parle alors de front-end navigator), ou bien, de plus en plus souvent, dans le SGBD contenant le cube (back-end navigator).

2.2.5 Les différentes stratégies : état de l'art des méthodes

Il existe de nombreuses stratégies de mise à jour de cube. Le temps est l'élément le plus critique ; la stratégie adoptée sera fonction des contraintes temporelles au niveau des sources (disponibilité, fréquence des changements) et des limites de vitesse du SGBD contenant le cube [Bouzeghoub *et al.*, 1999]. La stratégie de mise à jour est une combinaison des deux éléments répondant respectivement aux questions : **comment** et **quand** effectuer la mise à jour, décrits dans le Tableau 2.2. Ce tableau est inspiré des travaux de Gupta et Mumick [1999b] détaillant les stratégies de maintenance des vues matérialisées, de Ponniah [2001] et Adamson [2006] qui comparent la reconstruction totale avec la méthode incrémentielle et de Lambert [2006] qui propose de peupler des partitions journalières à intégrer au cube ensuite.

Le choix d'une stratégie de mise à jour dépend des limites du système et des besoins. On choisira une stratégie de mise à jour immédiate si l'application a des besoins temps réel ou proches du temps réel, mais une telle stratégie est difficile à implanter. De plus, certains éléments sont incompatibles : en général, il est impossible de reconstruire tout le cube immédiatement, car la reconstruction totale est souvent très coûteuse en temps de calcul.

Question	Stratégie		Explication
Comment ?	**Reconstruction totale**		Suppression du cube entier et reconstruction à partir de toutes les données archivées, la nuit en général.
	Incrémentielle		Intégration des mises à jour uniquement et recalcul des agrégats par incréments, calcul récursif.
	Pseudo-incrémentielle		Intégration des mises à jour uniquement mais reconstruction entière des agrégations par propagation (explication détaillée en 3.3.3).
	Partitionnement		Peuplement du cube par partitions en adoptant une des méthodes ci-dessus pour peupler les partitions
Quand ?	**Immédiate**		Déclenchée lors de la détection d'un changement au niveau des sources, la mise à jour est déclenchée par les systèmes source (PUSH). Automatique.
	Différée	*paresseuse (lazy)*	Lancée juste avant l'exécution de la requête d'un usager. L'information est toujours à jour pour la requête. Automatique.
		délai forcé	La mise à jour est déclenchée après un certain nombre seuil de modifications sur les sources, ou l'atteinte d'une valeur seuil pour les agrégats. Automatique.
		pré-établie	Programmée à l'avance à certaines dates, souvent périodiquement. Automatique.
		à la demande	À la demande d'un administrateur, par exécution de procédure ETL ou par une autre application (service web, etc.). Manuel.

TAB. 2.2 – Etat de l'art des stratégies de mise à jour de cube.

Pour mettre en place une stratégie de mise à jour, il faut adopter des techniques propres à la stratégie choisie pour chaque étape du processus. Les prochaines sous-sections présentent les différentes techniques pour les étapes les plus complexes soit : l'extraction des mises à jour, leur intégration directe dans le cube et le rafraîchissement des agrégations, en se focalisant sur la méthode incrémentielle afin de réduire le nombre

de calculs nécessaires et de garder un historique des modifications. Le choix de la méthode incrémentielle a été établi dès le début de la recherche, comme nous l'avons énoncé dans ce mémoire dans le chapitre 1. Elle permettrait entre autres de réduire le temps nécessaire à la mise à jour du cube, d'envisager de plus gros cubes de données avec des mesures spatiales complexes et de se tourner vers des applications temps réel.

2.3 Problèmes liés à la mise à jour dans un contexte décisionnel

2.3.1 Les problèmes liés à la mise à jour de cube

Plusieurs problèmes sont à la source de la complexité du processus :

- asynchronisme et processus parallèles
- fréquente évolution du processus
- nombreuses contraintes imposées par différents acteurs de l'entrepôt de données (utilisateurs, administrateurs de l'entrepôt, administrateurs des sources de données) respectivement en termes de besoins, de limite d'espace de stockage, d'accès aux données sources, etc.

Il n'existe donc pas une simple et unique stratégie de mise à jour. Les éléments jouant un rôle dans la mise à jour du cube sont très nombreux et il serait impossible d'en établir la liste complète. On peut cependant les classer en deux catégories : les éléments internes au cube et les éléments externes.

Parmi les éléments **internes** on compte essentiellement la modélisation employée pour le cube. « La cohérence de la méthode de modélisation est la clé pour des architectures reproductibles, à plusieurs échelles, utilisables et gérables. » traduit de l'anglais à partir de [Kimball et Caserta, 2004]. Les processus de mise à jour vont différer selon les différents types d'implantation possibles des dimensions et de la table de faits du cube. Dans ce mémoire, nous travaillons avec une implantation en schéma étoile, et ne détaillons pas les autres. Le lecteur est invité à se référer à [Kimball et Caserta, 2004] pour plus de

précisions. La méthode choisie pour implanter les agrégations influence également le processus. On recense deux modèles principaux : agrégats et données détaillées dans une même table ou agrégats et données détaillées dans des tables différentes. Dans le premier cas, la mise à jour devra effectuer plusieurs accès en lecture et écriture dans une même table, ce qui est très coûteux en temps car l'exécution de chaque étape de mise à jour des faits (détaillés et agrégés) doit attendre la fin de l'exécution de l'étape précédente.

Les éléments **externes** au cube (mais internes à l'entrepôt) sont les tâches périphériques à la mise à jour des cubes, à savoir :

- la méthode de mise à jour : reconstruction, pseudo-incrémentielle ou incrémentielle. La plus difficile à mettre en oeuvre est la méthode incrémentielle, car elle réserve un grand défi pour l'ETL : parvenir à extraire uniquement ce qui a changé. En effet, lors du peuplement initial, l'ETL ne se préoccupe que d'un seul jeu de données alors que pendant la mise à jour incrémentielle, l'ETL dispose de deux jeux de données : les données du cube et les données sources, desquelles il doit extraire uniquement les mises à jour. Plusieurs techniques existent pour repérer les mises à jour, elles seront abordées en 3.1. L'extraction des changements représente un défi de taille pour les données spatiales car peu de jeux de données spatiales transactionnelles gardent cette finesse d'information [Badard, 2000; Pouliot *et al.*, 2004].
- le déclenchement de la mise à jour (scheduling). Le processus peut être appelé par le cube (PULL) et se fait par lots : à date précise, périodiquement ou bien selon des événements comme le dépassement d'un seuil, etc. Dans l'autre cas, la mise à jour peut être commandée par les sources (PUSH) et se fait en flux ou par lots.
- l'archivage des cubes.

L'aspect temporel est primordial, la mise à jour est consommatrice de ressources et les travaux de recherche effectués pour optimiser le processus ont souvent pour objectif de réduire le temps requis afin de migrer vers des applications temps-réel. Le temps est présent à plusieurs niveaux, notamment dans les aspects de performance globale et dans

les tâches de programmation (déclenchement), ces deux dernières étant évidemment liées.

2.3.2 Les problèmes propres aux cubes de données spatiales

Les recherches effectuées sur les entrepôts de données spatiales se sont jusqu'à présent essentiellement concentrées sur la définition des concepts et des caractéristiques des systèmes SOLAP [Rivest *et al.*, 2001; 2005a] et sur la modélisation des cubes de données spatiales [Marchand *et al.*, 2004; Malinowski et Zimányi, 2004]. Aujourd'hui elles se tournent plus vers l'agrégation spatiale [Kuijpers et Vaisman, 2007; Raffaetà *et al.*, 2007; Gomez *et al.*, 2007; Grenier, 2007] dans le but d'exploiter toutes les capacités d'analyse qu'offriraient les systèmes SOLAP en intégrant des opérateurs d'agrégation pour les mesures spatiales géométriques [Bimonte *et al.*, 2006; Gomez *et al.*, 2007].
Ainsi peu de recherches se sont intéressées à la mise à jour des cubes spatiaux. Le processus est très complexe, car intègre à la fois les problèmes liés à la mise à jour des bases de données à référence spatiale (2.1) et les problèmes énoncés en 2.3.1 concernant la mise à jour des cubes de données non spatiaux. Nous reviendrons plus en détail sur les spécificités des cubes de données spatiales au 4.1, nous exposons ici rapidement les difficultés que devra surmonter une procédure de mise à jour de cube spatial. Elles sont d'ordres technologique et conceptuel.
Premièrement, du point de vue conceptuel on recense les obstacles suivants :

- aucune méthode n'a été formulée pour décrire le processus de mise à jour de cube non spatial ni de cube spatial.
- la mise à jour des mesures spatiales géométriques est impossible sans opérateurs d'agrégation spatiale.
- il existe différentes implantations possibles pour inclure la géométrie au sein des dimensions et des faits
- les contraintes d'intégrité ne sont pas toutes connues [Salehi *et al.*, 2007]
- il existe peu de méthodes pour implémenter les données raster [McHugh, 2008].
- il faut tenir compte du processus de généralisation cartographique déjà complexe

pour les bases de données à référence spatiale [Frigioni et Tarantino, 2003; Haunert et Wolff, 2006; Sabo et Bédard, 2007; Nakos *et al.*, 2008; Mackaness et Chaudhry, 2008] et du problème de mismatch entre la généralisation cartographique et l'agrégation multidimentionnelle [Bédard *et al.*, 2006].

Deuxièmement, du point de vue technique, les obstacles sont également nombreux. Tout d'abord, il n'existe pas à l'heure actuelle de véritable **ETL multidimensionnel** gérant les données spatiales, un prototype s'appuyant sur le logiciel ETL Kettle de la société Pentaho[6] a été mis au point au Centre de Recherche en Géomatique de l'Université Laval au sein de l'équipe GeoSOA, dirigée par le Dr. Thierry Badard : GeoKettle[7]. Il permet certaines opérations de base, mais reste en développement. Un outil ETL géo-décisionnel devra gérer les multiples formats de données spatiales existants et s'enrichir de nouveaux opérateurs dédiés aux cubes géo-décisionnels. Cette multitude de formats complexifie la gestion des éléments spatiaux au sein des dimensions, le plus simple serait d'adopter un même format pour les géométries dans le cube de données. En outre, si l'on désire stocker la géométrie dans une base de données, on dispose d'un choix croissant de SGBDs (PostGIS, Oracle Spatial, IBM...). Face à cette multitude de formats, la standardisation s'avère primordiale. Le développements d'applications et de logiciels géomatiques (p. ex. FME) est désormais guidé par les nombreux standards ISO TC/211 et par les standards OGC comme le Simple Feature Specification adopté par les SGBD énoncés ci-dessus. Malgré les récents progrès en matière d'interopérabilité et d'échange de données spatiales [Dubé *et al.*, 2007b], la gestion des différents formats de données reste complexe et pose des problèmes de droits d'auteurs. Ensuite, les données spatiales sont plus complexes que les données ordinaires : leur traitement nécessite plus de temps de calcul, elles sont très gourmandes en espace mémoire d'où le besoin de trouver des algorithmes de sélection pour la matérialisation des agrégations [Stefanovic *et al.*, 2000]. La géométrie contient non seulement des informations explicites destinées à la représentation cartographique, mais également de nombreuses informations implicites, notamment concernant la sémantique liée à la forme et à la position (p. ex. un triangle

[6] http ://kettle.pentaho.org

[7] http ://www.geokettle.org

représente rarement une route mais plutôt un bâtiment, etc.).

2.4 Conclusion du chapitre

Ce chapitre nous entraîne directement dans l'univers des mises à jour de données. Nous avons d'abord exposé brièvement les aspects du processus de mise à jour de données dans les bases de données transactionnelles. Le sujet de recherche s'ancre dans un processus plus large : la maintenance des entrepôts de données, que nous avons explicité. Une typologie de maintenance d'un entrepôt de données, ainsi qu'une typologie détaillée de la *mise à jour de cube* sont fournies. Les étapes du processus de mise à jour de cube ont été expliquées en un état de l'art des différentes stratégies de mises à jour de cube a été donné. Ces considérations nous ont permis de prendre conscience des différentes facettes de la mise à jour dans un contexte décisionnel et de les comparer avec le contexte transactionnel (comparaison résumée dans le Tableau 2.3). Il en ressort que les bases de données transactionnelles sont destinées à subir des mises à jour à une fréquence élevée puisque chaque transaction enregistrée dans le système déclenche une mise à jour. Nous observons que la problématique a largement été traitée dans la littérature concernant le domaine transactionnel, à commencer par les bases de données classiques où des opérateurs spécifiques ont été intégrés au langage standard, puis dans les bases de données à référence spatiale avec les techniques spécialisées comme l'appariement de données, etc [Badard, 2000; Pouliot *et al.*, 2004]. Les complications liées à l'ajout de la temporalité ont été surmontées par les techniques de versionnement. Il est en tout autrement dans le domaine décisionnel ; une confusion règne entre les auteurs et les travaux existants. Le processus est tellement complexe que les recherches n'en abordent souvent qu'une petite partie et quelques solutions sont proposées pour les entrepôts non spatiaux. Les entrepôts spatiaux apportent de nouveaux problèmes dont la résolution passe par la définition de nouveaux concepts.

Mise à jour	Bases de données transactionnelles	Bases de données décisionnelles
Complexité du processus	simple : peu d'étapes	complexe : nombreuses étapes
Temps requis	court	long
Déclenchement	immédiat, temps réel	souvent différée
Fréquence	opération courante	peu fréquentes
Gérée par	Système transactionnel	outil ETL
Modifications possibles	ajout, correction, suppression	uniquement des ajouts
Langage standard dédié	SQL (insert, update, delete)	aucun
Historique	souvent aucun, données courantes	présent, données courantes et historiques
Méthodes	souvent incrémentiel	nombreuses méthodes (reconstruction totale, pseudo-incrémentielle, incrémentielle, etc.)
Propagation	entre bases de données	en interne
Typologie pour les mises à jour de données spatiales	existe	aucune
Problèmes liés aux données spatiales	connus, solutions existantes	peu connus

Tab. 2.3 – La mise à jour dans les contextes transactionnel et décisionnel

Chapitre 3

Méthodes existantes pour la mise à jour de cubes non spatiaux

Ce chapitre pose les prémisses de la définition de méthodes de *mise à jour de cube de données spatiales*, en étudiant et expliquant les processus et méthodes existants dans le domaine des entrepôts de données non spatiales. Nous avons vu au chapitre 2 (cf. 2.3.1) que le processus de mise à jour est un processus complexe, mettant en jeu plusieurs éléments. Un état de l'art des techniques employées est présenté pour chaque étape du processus de mise à jour de cube : l'extraction des mises à jour (section 3.1), leur intégration dans le cube (section 3.2), et le rafraîchissement des agrégations (section 3.3). Le chapitre clôt sur la nécessité de définir de nouvelles méthodes de mise à jour pour les cubes de données spatiales et ébauche un ensemble de constatations à prendre en considération pour la suite (section 3.4).

3.1 L'extraction des mises à jour au niveau des sources

Durant le peuplement initial du cube, la détection des changements sur les données sources n'est pas une étape cruciale, mais une fois complétée, elle devient une étape primordiale. Pour mettre à jour le cube, il est préférable d'adopter une méthode

incrémentielle, vient alors le vrai challenge pour l'ETL : réussir à capturer uniquement les données qui ont changé et non toutes les données des sources. Les techniques d'extraction des changements dépendent des types de sources de données et de l'effort de travail que l'équipe est prête à déployer. Ces techniques peuvent être classées en deux catégories : extraction immédiate et extraction différée. Une classification synthétique de toutes les techniques énoncées est fournie dans le Tableau 3.1. Elle est inspirée de [Ponniah, 2001; Darmawikarta, 2007; Kimball et Caserta, 2004].

	Quand ?	
	Immédiate	**Différée**
Livraison des données	flux	par lots
Qui ?	PUSH (sources)	PULL (ETL)
Reconstruction		tel quel (aussi pour le peuplement initial)
Incrémentiel	logs de transaction au niveau des sources	timestamp
Pseudo-incrémentiel	déclencheurs (triggers) dans la base de données source	dimension d'audit
	capture des changements dans les applications source	comparaison totale avec la copie d'extraction précédente

TAB. 3.1 – État de l'art des techniques d'extraction de données.

La livraison des données extraites peut se faire de manière continue, en flux ou bien discontinue, par lots. Le déclenchement du processus est soit initié par les données sources (PUSH) dans le cas d'un flux continu de données, soit par l'ETL (PULL) dans le cas d'une mise à jour par lots. Les techniques employées pour extraire les données sont propres à la méthode de mise à jour (Tab. 2.2) et du type d'extraction (immédiate ou différée). La reconstruction totale du cube nécessite l'extraction de toutes les données source et ne peut être effectuée immédiatement. Pour les méthodes incrémentielle et pseudo-incrémentielle, seules les mises à jour sont extraites. L'extraction immédiate ne peut être gérée que par les sources, seules aptes à relever en direct les modifications. Cela peut être effectué par des logs, des déclencheurs ou bien par un méthode spécifique à l'application source. La détection des modifications en différé se fait par comparaison. Elle peut être effectuée à l'aide de *timestamp* ajoutés dans les données sources par

comparaison aux dates de mise à jour stockées dans une dimension d'audit. La méthode la plus coûteuse et la moins complexe à mettre en oeuvre reste la comparaison totale des sources actuelles avec les sources avant la dernière mise à jour.

3.2 Les techniques pour la mise à jour des dimensions

Beaucoup de travaux de recherches se sont concentrés sur l'évolution de la structure des dimensions [Blaschka *et al.*, 1999; Vaisman, 2001; Bédard *et al.*, 2003; Bakillah, 2007] et proposent des opérateurs d'évolution structurelle. Rageul [2007] introduit le concept de révisions des données dans le cadre de la fouille archéologique : l'archéologue modifie la structure des dimensions afin de mieux interpréter les résultats. L'inconvénient de cette dernière approche est l'impossibilité de garder l'historique des modifications.
Seuls Kimball et Ross [2002] proposent des techniques simples et efficaces pour la mise à jour des données dans les dimensions, au sens de la typologie de mise à jour de cube que nous avons énoncée en 2.2.3.2. Ces techniques ont d'ailleurs été implantées dans plusieurs outils ETL (notamment l'outil Open Source Kettle de la société Pentaho) et sont très connues dans la sphère des entrepôts de données. Nous les exposons dans la suite de cette section.
Kimball et Ross [2002] classifient les dimensions en deux catégories principales : celles qui varient lentement (la plupart des dimensions) et celles qui varient rapidement. Les premières sont qualifiées de **Slowly Changing Dimensions** (l'acronyme *SCD* est relativement connu dans le monde du business intelligence) et les secondes, de **Rapidly Changing Dimensions** (*RCD*).

3.2.1 Les Slowly Changing Dimensions

La plupart des dimensions sont souvent considérées comme statiques comparées aux tables de faits, mais leurs contenus ne sont pas pour autant figés ; ils évoluent

plus lentement. Kimball suggère trois techniques pour mettre à jour les attributs des dimensions *SCDs* : les techniques de type 1, de type 2 et de type 3 ; certaines permettent de garder un historique des modifications et d'autres ne le permettent pas.

Ces techniques seront détaillées par la suite, mais avant, il est nécessaire d'expliquer un peu de théorie sur les colonnes de dimension. Il existe principalement six types de colonnes dans une table de dimension :

- ***SK*** : *Surrogate Key*, clé primaire dans la dimension, elle ne doit avoir aucune signification particulière. C'est la *SK* qui lie la dimension à la table de faits détaillés.
- ***NK*** : *Natural Key*, identifiant naturel de l'élément dans les sources transactionnelles (p. ex. : le code de province *QC* pour le Québec).
- ***SCD1*** : attribut mis à jour suivant la technique de type 1.
- ***SCD2*** : attribut mis à jour suivant la technique de type 2.
- ***SCD3*** : attribut mis à jour suivant la technique de type 3.
- ***HK*** : *Housekeeping columns*, colonnes utiles pour l'entretien du cube, pour savoir ce qui a changé (p. ex. : les colonnes de dates de validité de l'occurrence, tous les attributs d'une dimension d'audit, etc.).

La technique *SCD 1* (pour type 1) consiste à remplacer directement l'ancienne valeur de l'attribut avec sa nouvelle valeur (comme illustré en Figure 3.1). L'historique des modifications n'est pas préservé.

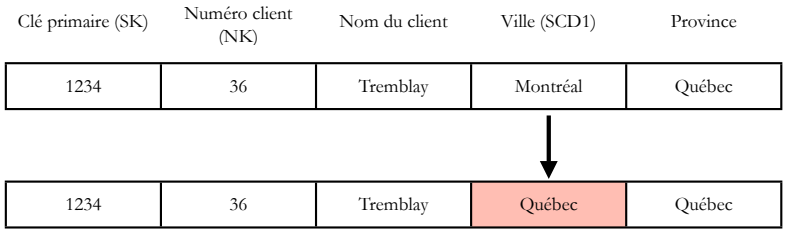


FIG. 3.1 – Mise à jour d'une Slowly Changing Dimension de type 1.

Les techniques *SCD 3* et *SCD 2* sont basées sur le versionnement d'occurrence. Nous expliquons d'abord brièvement le type 3 pour détailler ensuite le type 2, qui est de notre point de vue, la méthode la plus puissante pour mettre à jour une dimension ne

subissant pas de mises à jour fréquemment.

La technique *SCD 3* est utilisée quand un changement se produit, mais que l'on désire garder l'ancienne valeur en second choix. Elle consiste à ajouter une colonne « ancienne valeur » et à remplacer la valeur courante, comme illustré en Figure 3.2. L'historique des modifications n'est pas entièrement préservé, puisque l'on ne dispose que de l'ancienne valeur.

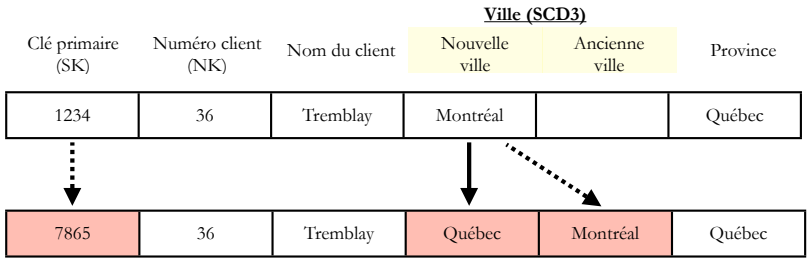


FIG. 3.2 – Mise à jour d'une Slowly Changing Dimension de type 3.

La technique *SCD 2*, illustrée en Figure 3.3, consiste à créer une nouvelle occurrence du membre avec la nouvelle valeur de l'attribut modifié. Chaque occurrence possède une période de validité matérialisée souvent par deux colonnes de date : *date de début* et *date de fin*.

Le but de l'entrepôt de données est d'entreposer les données historiques et courantes à des fins d'analyse. La technique de type 2 répond à cette exigence, car elle partitionne implicitement l'historique (Fig. 3.4) et, de ce fait, c'est la technique prédominante dans le monde des entrepôts de données, mais elle requiert un bon système d'extraction capturant uniquement les changements.

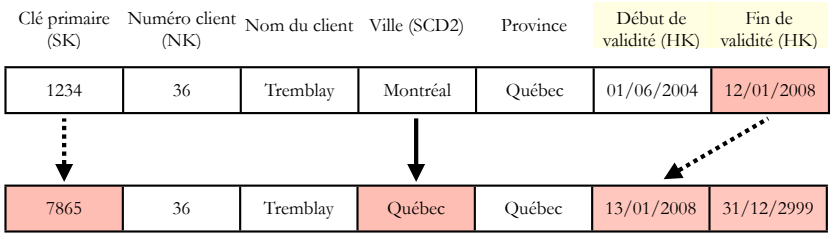


FIG. 3.3 – Mise à jour d'une Slowly Changing Dimension de type 2.

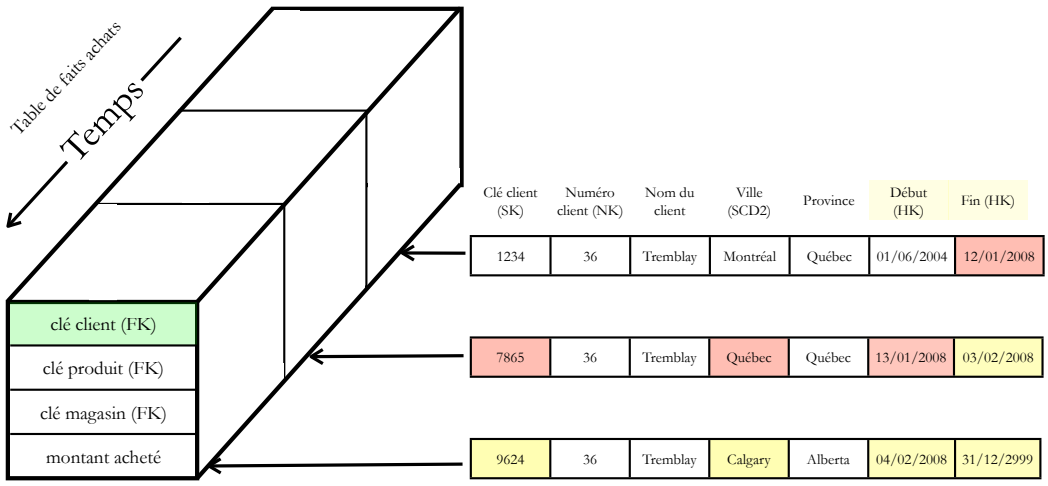


FIG. 3.4 – Partitionnement de l'historique avec les *SCD2*.

Inspiré de Kimball et Caserta [2004].

3.2.2 Les Rapidly Changing Dimensions

La technique des *Slowly Changing Dimensions de type 2* est adaptée à la plupart des dimensions, mais pose des problèmes s'il s'agit d'une « grosse dimension » variant assez souvent. C'est le cas des dimensions *client* d'une banque ou d'un grand magasin, etc., mais aussi des dimensions dans les applications temps réel. Elles peuvent dépasser les millions d'enregistrements et il devient difficile de gérer les mises à jour par la technique *SCD2*, puisque la taille de la dimension exploserait. Kimball et Ross [2002] proposent une technique spéciale pour ces types de dimensions qu'il qualifient de *Rapidly Changing Monster Dimensions* : la scinder en deux parties, une dimension *SCD* « assez grosse » et une *minidimension* fixe. La dimension *RCD*, réunion des deux précédentes, possède deux clés étrangères dans la table de faits, la mise à jour se fait directement par changement d'association de clés dans la table de faits. Cette technique partitionne l'historique de manière automatique au sein de la table de faits. Un exemple est donné en Figure 3.5 : *le client Charles Tremblay vient de fêter ses 20 ans le 23 janvier 2008, son âge doit être mis à jour dans la dimension client.* Chaque cadre montre l'exécution de la mise à jour. Dans le premier cadre, on adopte la technique *SCD2* : la mise à jour se manifeste par l'ajout d'une rangée dans la dimension et le changement des dates de validité de l'ancienne occurence. Dans le second, on adopte la technique *RCD*, la

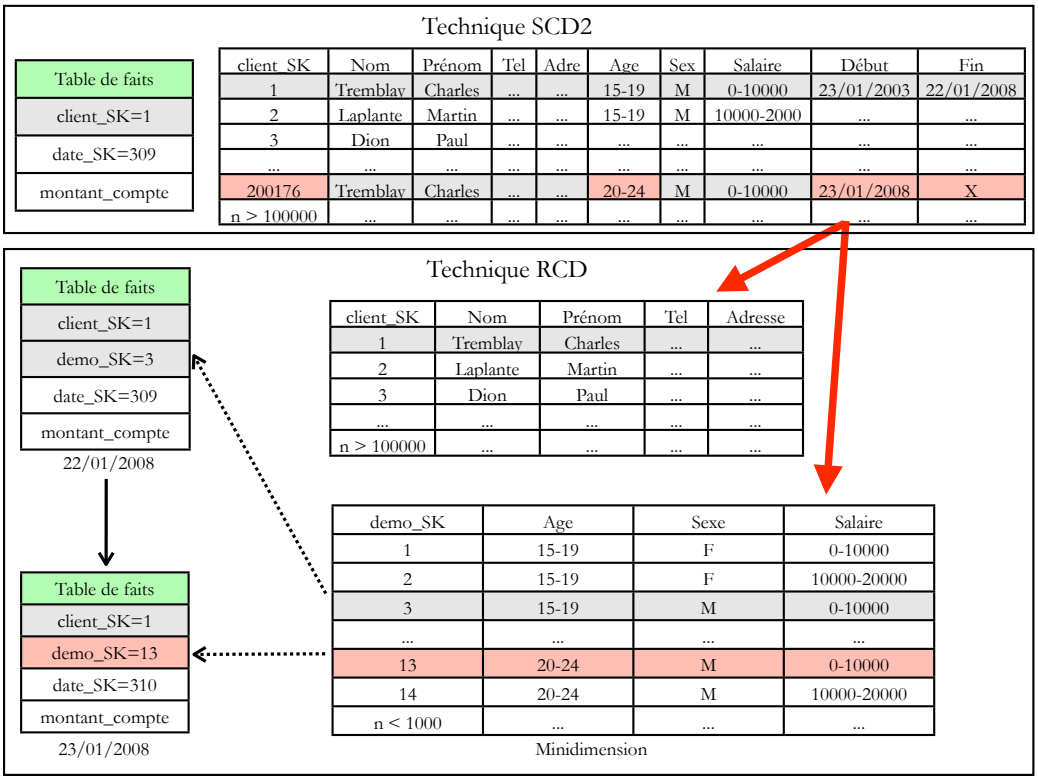


FIG. 3.5 – Exemple de passage d'une technique SCD2 à RCD.

mise à jour se manifeste par l'ajout d'une rangée dans la table de faits, avec une clé démographique (*demo_SK*) différente. Les flèches montrent la division de la dimension client en deux dimensions : une dimension *SCD* pour les données personnelles et une *minidimension* contenant des intervalles pour les données démographiques.

3.3 Les techniques pour la mise à jour des agrégats

3.3.1 L'agrégation dans les cubes de données

Les agrégations sont des mesures résumées, d'une granularité supérieure aux mesures détaillées, regroupées à l'aide d'un opérateur d'agrégation. Si un cube comprend comme mesure de fait détaillés : *la population du Canada, par âge et par langue maternelle*, un exemple d'agrégat pourrait être : *le nombre de Canadiens âgés de moins de 20 ans et dont le français est la langue maternelle.* Il existe de nombreux

opérateurs d'agrégation, les plus utilisés sont les opérateurs SQL : COUNT(), COUNT(distinct), SUM(), MIN(), MAX() et AVG().

On peut classer les fonctions d'agrégations selon trois types [Gray *et al.*, 1997] :

* Distributive : Une fonction d'agrégation $F()$ est distributive s'il existe une fonction $G()$ telle que $F(\{X_{i,j}\}) = G(\{F(\{X_{i,j}\}|i = 1, ..., I\}|j = 1, ..., J)$. COUNT(), SUM(), MIN() et MAX() sont distributives.
* Algébrique : Une fonction d'agrégation $F()$ est algébrique s'il existe une fonction $G()$ et une fonction $H()$ telles que $F(\{X_{i,j}\}) = H(\{G(\{X_{i,j}\}|i = 1, ..., I\}|j = 1, ..., J)$. AVG() est algébrique puisque composée de SUM() et de COUNT().
* Holistique : Une fonction d'agrégation est holistique s'il n'existe pas de constante M telle que $F(\{X_{i,j}\}|i = 1, ..., I\})$ puisse être caractérisé par une fonction sur M partitions. RANK() est un exemple de fonction holistique.

Les agrégations sont obtenues en appliquant un opérateur d'agrégation sur les mesures de la table de faits suivant certains niveaux des dimensions. Cependant, il n'est pas toujours possible d'agréger les mesures selon toutes les dimensions (*sommer les stocks d'un magasin sur plusieurs jours reviendrait à compter plusieurs fois les mêmes éléments*). Il existe trois types de mesures [Horner *et al.*, 2004] :

* additives : peuvent être agrégées selon toutes les dimensions
* semi-additives : peuvent être agrégées selon certaines dimensions. *Par exemple la somme des stocks de l'enseigne X peut être calculée en additionnant les stocks des différents magasins X (dimension magasin), mais pas selon le temps (dimension temporelle), ce qui reviendrait à compter plusieurs fois les mêmes éléments.*
* non-additives : ne peuvent pas être agrégées

3.3.2 Les différentes implantations

Il existe trois manières d'implanter les agrégations pour un cube, elle sont résumées dans [Adamson, 2006]. Nous les détaillons ici en se basant sur le cube représenté par le schéma étoile de la Figure 3.6. Les trois types d'agrégation sont :

1. **schéma unique** pour les faits détaillés et les agrégats : les faits agrégés et détaillés sont stockés dans la même table. Les clés étrangères pointent successivement vers plusieurs colonnes des dimensions (Fig. 3.7).
2. **agrégations préjointes** (*pre-joined aggregates*). Chaque agrégation est contenue dans une table comprenant également les membres de dimension (Fig. 3.8).
3. **agrégations en schéma étoile** (*star schema aggregates*). Les agrégats sont dans des tables de faits agrégés, liées à des dimensions agrégées (Fig. 3.9).

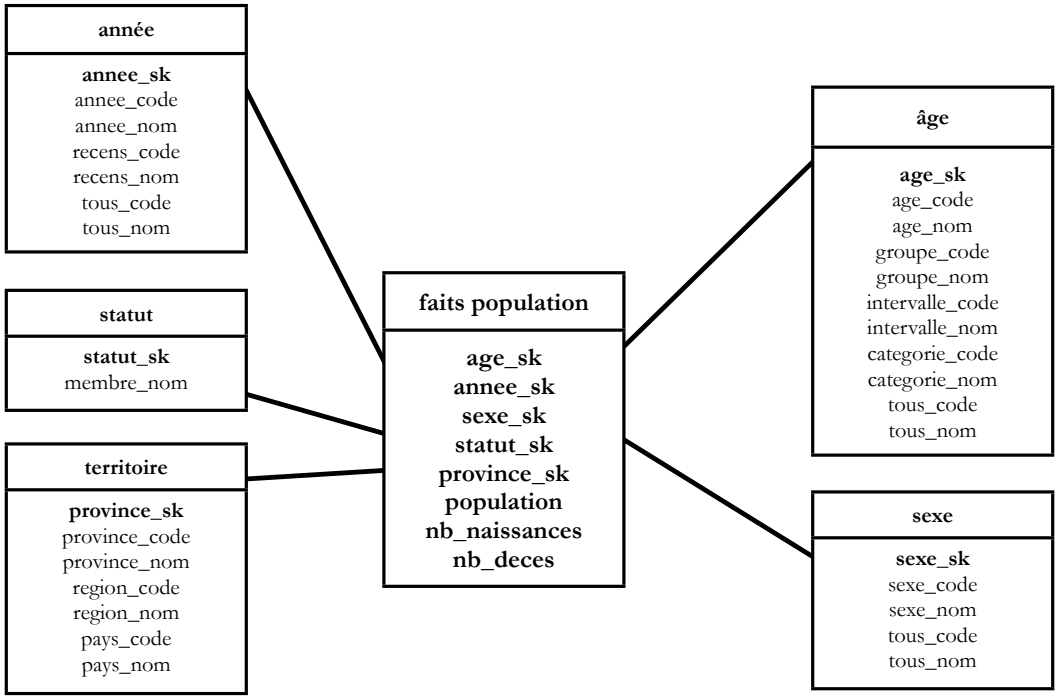


FIG. 3.6 – Schéma étoile du cube : faits détaillés.

Au niveau conceptuel, on peut représenter le cube selon une *lattice* contenant 2^n, noeuds ; avec n le nombre de dimensions du cube. On considère le cube *sales_orders* (présenté en Annexe B, Fig. C.1) constitué de trois dimensions : *product, customer,*

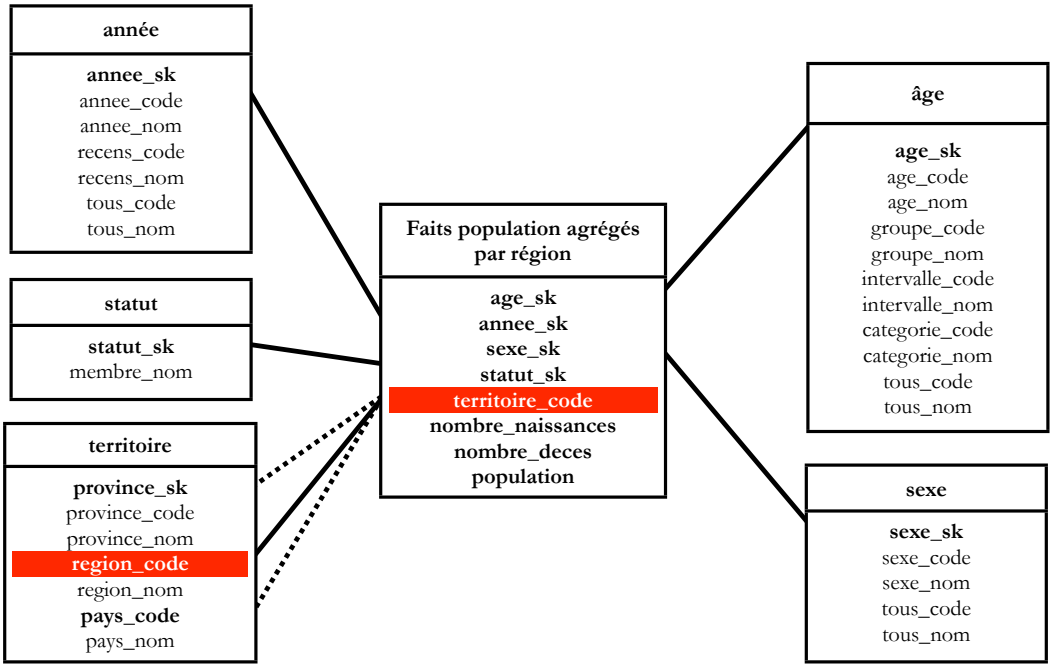


FIG. 3.7 – Faits détaillés et agrégés dans une même table.

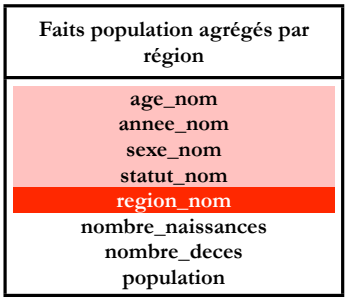


FIG. 3.8 – Agrégats préjoints.

date, il est représenté selon la lattice de la Figure 3.10.

Chaque hiérarchie de dimension peut également être symbolisée par une lattice (Fig. 3.11), et la combinaison des deux lattices donne une *lattice combinée* contenant toutes les agrégations possibles du cube : soit 60 agrégations ici (Fig. 3.12). Le nombre d'agrégats possibles (donc de noeuds de la lattice combinée) en absence de hiérarchies dans les dimensions est exprimé par :

$n_{noeuds} = \prod_i (2^{n_i} + 1)$, avec n_{noeuds} le nombre de noeuds donc d'agrégats possibles et n_i le nombre d'attributs de la dimension i .

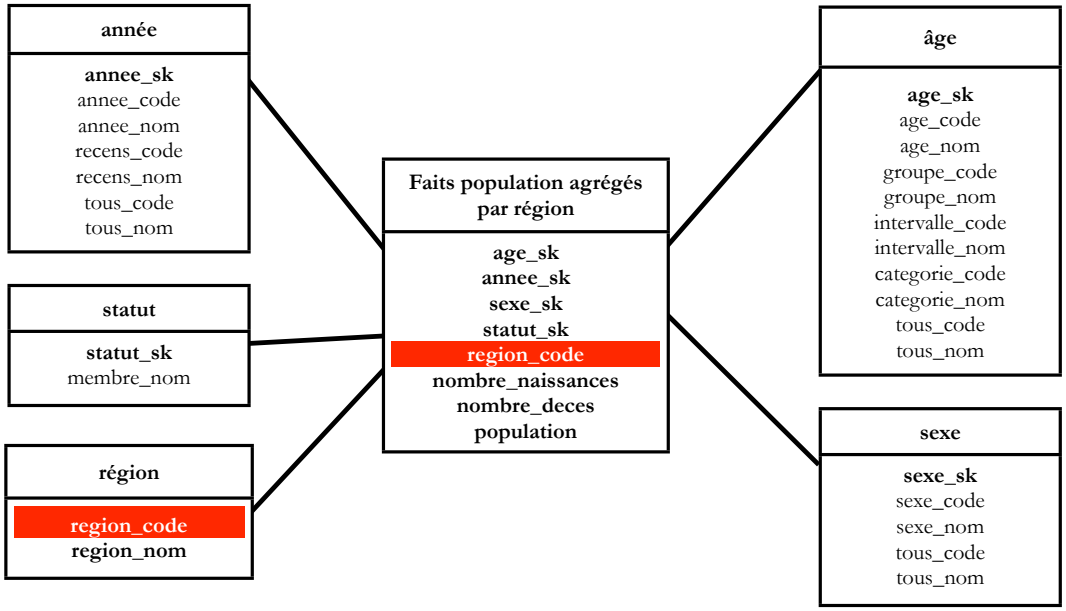


FIG. 3.9 – Agrégats en schéma étoile.

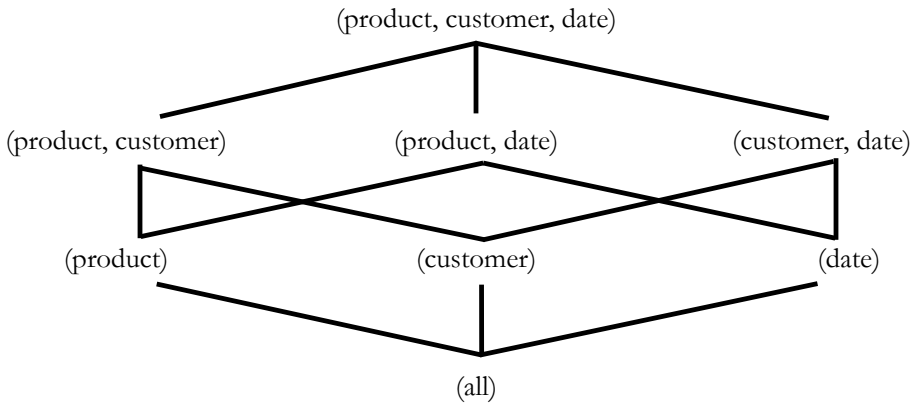


FIG. 3.10 – Lattice d'un cube de données.

Il est possible de précalculer et stocker les agrégations, afin d'accélérer les requêtes, plutôt que de réaliser leur calcul à la volée. Cependant, précalculer toutes les combinaisons possibles augmenterait le temps requis pour la maintenance des agrégations et donc pour la mise à jour de cube. Mettre à jour le cube signifie en effet mettre à jour toutes ses données, et donc, entre autres, mettre à jour tous les agrégats précalculés (matérialisés). Pour réduire le temps requis pour la mise à jour de cube, il est préférable de sélectionner les agrégations à matérialiser dans le cube (les agrégations les plus requêtées, utilisées) [Harinarayan *et al.*, 1996; Pendse, 2005a; Shukla *et al.*, 1996].

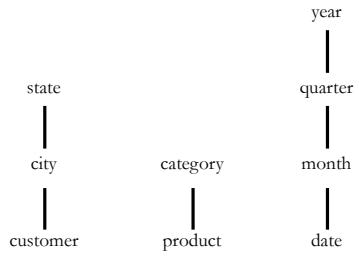


FIG. 3.11 – Lattice de dimensions.

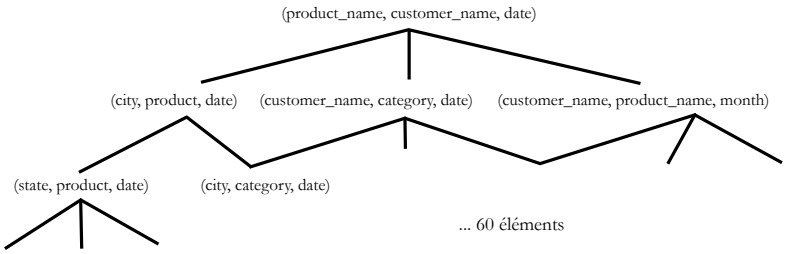


FIG. 3.12 – Lattice combinée.

Les SGBDs commerciaux offrent des techniques pour stocker les agrégats : les plus populaires sont les *Materialized Views* (*vues matérialisées*) chez Oracle, les *Materialized Query Tables* chez IBM, les *Automated Tables* (*tables automatisées*) chez RedBrick et les *Indexed Views* (*vues indexées*) chez SQL Server de Microsoft. Le principe est sensiblement le même et revient au concept de *vues matérialisées* auquel nous nous attachons ici.

Tout comme une vue relationnelle, une **vue matérialisée** est définie par une requête SQL SELECT, mais ses tuples sont stockés, matérialisés dans la base de données. Lorsqu'un utilisateur soumet une requête sur la table de faits détaillés, le navigateur d'agrégations réécrit la requête selon le schéma de la vue. Les vues matérialisées peuvent être perçues comme une sorte de cache accélérant les requêtes.

Un exemple de requête de création d'une vue matérialisée sous Oracle est donné en Figure 3.13 (cube Fig. C.4).

Les vues matérialisées présentent de nombreux avantages : accès rapide aux données, réduction de charge de la CPU, des coûts de stockage, etc. Elles permettent une réduction des coûts. Le bénéfice lié à leur utilisation est formalisé dans Rafanelli [2003] ; soient :

```
CREATE MATERIALIZED VIEWC_AGG_POP_PROV_AGcat_AN_R_MV
  BUILD IMMEDIATE REFRESH FAST ON DEMAND
AS SELECT age.CATEGORIE_NOM, annee.ANNEE_NOM, province.REGION_NOM, SUM(f.
    nb_naissance), SUM(f.nb_deces), SUM(f.population) population
FROM C_POP_PROVINCE_FACT f, C_AGE_DIM age, C_ANNEE_DIM annee, C_PROVINCE_DIM
    province
WHERE f.D_AGE_KEY = age.D_AGE_KEY
AND f.D_ANNEE_KEY = annee.D_ANNEE_KEY
AND f.PROVINCE_CODE = province.SK
GROUP BY age.CATEGORIE_NOM, annee.ANNEE_NOM,   province.REGION_NOM;
```

FIG. 3.13 – Exemple de requête de création d'une vue matérialisée sous Oracle 10g.

- Q_{DB}, le coût temporel lié à l'exécution de la requête pour une vue V
- Q_M, le coût temporel lié à la lecture de la vue matérialisée correspondante MV (en général $Q_{DB} > Q_M$)
- C_M, le coût de stockage de la vue matérialisée MV
- f_Q, la fréquence de requêtage
- f_U, la fréquence de recalcul complet de MV

Le bénéfice d'utiliser la vue matérialisée est exprimé par la différence entre le coût des requêtes sur la vue V ($f_Q \cdot Q_{DB}$) et le coût d'utilisation de la vue matérialisée. Ce dernier est égal à la somme des coûts de stockage (C_M), de mise à jour ($f_U \cdot Q_{DB}$) (cas du recalcul complet) et de requêtage ($f_Q \cdot Q_M$).

Donc le bénéfice B est :

$B = f_Q \cdot Q_{DB} - (C_M + f_U \cdot Q_{DB} + f_Q \cdot Q_M)$

Plus il y a de requêtes sur la vue, plus il est bénéfique de la matérialiser, et plus la fréquence de mise à jour augmente, moins il est bénéfique de la matérialiser.

Matérialiser les agrégations permet donc d'accélérer le temps de requêtage. De plus, des méthodes ont été définies pour mettre à jour les vues matérialisées de manière incrémentielle (paragraphe suivant). Nous souhaitons donc explorer la technique des vues matérialisées pour stocker nos agrégations. Dans la suite du mémoire, les agrégations seront des agrégations préjointes sous forme de vues matérialisées et nous emploierons le terme *vues* pour désigner les vues matérialisées.

Les agrégats préjoints sont la forme d'agrégation qui s'accorde le mieux avec la technique des vues matérialisées [Adamson, 2006] : en effet, chaque type d'agrégat correspond à une table qui est également le résultat de la requête de définition de la vue. Stocker chaque agrégation dans une table différente (comme c'est le cas pour les agrégats préjoints) facilite la mise à jour, car divise le processus en la mise à jour de plusieurs tables dépendantes (cf. 3.3.3.3) et accélère les requêtes, car la lecture des enregistrements (guidée par le navigateur d'agrégations) se fait dans de plus petites tables. Agréger les dimensions augmente le nombre de tables, aussi nous n'avons pas opté pour la technique d'agrégats en schéma étoile. Bien que nous ayons choisi d'utiliser des agrégats préjoints sous forme de vues matérialisées pour nos explications et nos tests, nos méthodes sont conceptuelles et donc indépendantes de l'implantation choisie pour le cube.

3.3.3 La maintenance des agrégations

Recalculer entièrement les agrégats suite aux mises à jour quotidiennes est la méthode la plus simple à mettre en oeuvre, mais elle présente de nombreux inconvénients (cf. 1.1). Aujourd'hui la technique alternative de mise à jour incrémentielle est assez répandue dans les outils commerciaux. On distingue deux types de techniques incrémentielles en fonction du lieu de stockage des agrégats :

- les cubes de données (et donc les agrégations) sont dans un serveur MOLAP (*front-end*) : ce sont des cubes MOLAP (Multidimensional OLAP), stockés dans un format propriétaire, leur rafraîchissement se fait à l'aide de techniques propriétaires au serveur. Ce ne sont pas des algorithmes génériques[1].
- les cubes de données (et donc les agrégations) sont dans l'entrepôt ou dans un serveur ROLAP (Relational OLAP), sous forme de tables relationnelles (SGBD) (*back-end*). Ces tables seront soit de simples tables relationnelles mises à jour par des techniques ad hoc à l'application, ou bien des vues matérialisées (ou vues similaires) et leur rafraîchissement se fait à l'aide d'algorithmes génériques définis

[1] Ici : conceptuels et applicables à d'autres serveurs

pour les vues matérialisées.

La maintenance des agrégations se résume souvent (en pratique) au problème de maintenance des vues matérialisées.

3.3.3.1 Définitions

Les définitions suivantes sont données dans le cadre des bases de données et ne sont pas spécifiques aux entrepôts de données[2]. Nous employons le terme *vues* pour désigner les vues matérialisées.

La maintenance d'une vue, telle que définie par Gupta et Mumick [1999a], est le processus de mise à jour d'une vue suite aux changements qui ont eu lieu sur les données de base (sur lesquelles est construite la vue).

La maintenance incrémentielle d'une vue désigne le processus de maintenance de vue qui applique uniquement les changements ayant lieu sur les données de base. Il est souvent avantageux d'utiliser la maintenance incrémentielle : recalculer la vue en partant de zéro lorsque seule une petite partie de la vue change, est une perte de temps [Gupta et Mumick, 1999a].

Les travaux de recherche sur la maintenance des vues se sont surtout concentrés sur la maintenance incrémentielle ; de nombreux groupes de recherche universitaires et de recherche et développement dans les entreprises produisant des SGBDs se sont penchés sur ce problème entre 1977 et 2000 [Shmueli et Itai, 1984; Blakeley *et al.*, 1986; Gupta *et al.*, 1993; Griffin et Libkin, 1995; Mumick *et al.*, 1997] (pour n'en citer que les plus connus). En 2000, le sujet avait été largement abordé, le problème de maintenance incrémentielle des vues a trouvé de nombreuses solutions et les recherches s'y sont moins intéressées.

[2]Puisque les vues matérialisées ne sont pas uniquement utilisées dans le cadre des entrepôts de données.

3.3.3.2 Coût temporel de la mise à jour des vues

L'avantage des vues matérialisées prend toute son ampleur avec les techniques de **maintenance incrémentielle**, qui réduisent énormément les coûts de mise à jour des vues.

Soit C_U le coût temporel de mise à jour incrémentielle de la vue matérialisée. C_U est une petite fraction de Q_{DB} (coût lié recalcul au complet de la vue), puisque C_U implique l'ajout, la modification ou la suppression de petites parties de la vue matérialisée. Donc $C_U < Q_{DB}$, ce qui signifie que la mettre à jour la vue de manière incrémentielle est moins coûteux en temps que mettre la jour la vue en la recalculant complètement[3].

3.3.3.3 Solutions pour la maintenance des vues matérialisées

De nombreux algorithmes de maintenance des vues ont été proposés et la plupart sont regroupés dans [Gupta et Mumick, 1999a]. La contribution la plus importante dans le domaine des entrepôts de données OLAP reste l'algorithme défini par Mumick *et al.* [1997] : *Summary-delta table*, qui fut référencé ensuite par maints travaux sur la maintenance incrémentielle des vues. Mumick *et al.* [1997] divisent la maintenance en deux étapes : la propagation des mises à jour dans une vue matérialisée temporaire[4] (propagation) et l'insertion de ces mises à jour dans la vue finale (rafraîchissement). L'algorithme a deux objectifs : maintenir une vue dans le cadre de mises à jour par lots et maintenir un ensemble de vues qui sont interconnectées (définies sur la même table de faits).

Nous explicitons la méthode du **Summary-delta** pour l'insertion et la suppression de tuples[5] pour les exemples de vues matérialisées décrites en Figure 3.14 et créés à l'aide des codes SQL présentés en Figure 3.15 (cube illustré en Annexe B, Figures C.4 et C.6).

La première vue comporte les mesures de population agrégées selon la dimension *annee*,

[3] Ceci pouvait paraître intuitif, mais nous avons souhaité l'expliciter.

[4] Cette vue est appelée *summary delta table*, elle est de même structure que la vue matérialisée finale et contient uniquement les changements à appliquer.

[5] Bien que nous ne prenions pas en compte les suppressions dans notre typologie

d_annee_key	d_sexe_key	d_statut_pop_key	province_sk	population	cnt_all
1	1	1	1	16 800 000	200
1	1	2	1	200 000	29
1	2	1	1	16 900 000	150
1	2	2	1	190 000	21
2	1	1	1	17 100 000	33 000
2	1	2	1	30 000	43
2	2	1	1	17 100 000	32000
2	2	2	1	35 000	42

Can_population_annee_sexe_statut_province

d_annee_key	d_sexe_key	population
1	1	17 000 000
1	2	17 090 000
2	1	17 130 000
2	2	17 135 000

Can_population_annee_sexe

population
34 177 500

Can_population_annee

FIG. 3.14 – Exemples de vues matérialisées.

sexe, *statut* et *province* ; la deuxième vue, selon les dimensions *annee* et *sexe* et la troisième vue, selon la dimension *annee*. Chaque vue peut être déduite de la précédente, ce qui va permettre une mise à jour en cascade.

L'algorithme du **Summary-delta** est divisé en deux parties :

1. **propagation**[6] : en dehors de la fenêtre temporelle de mise à jour. Crée les tables *summary delta* (la vue est encore disponible pour répondre aux requêtes des utilisateurs).
2. **rafraîchissement**[6] : applique les changements inclus dans la table *summary delta* sur la vue (la vue n'est plus accessible).

La phase de propagation (Fig. 3.16) consiste à préparer les changements en calculant la table *summary delta*. Cette table est une table de différentiel qui va contenir uniquement les mises à jour agrégées (groupées selon la définition des agrégats dans la

[6]Traduction littérale de l'anglais à partir de [Mumick *et al.*, 1997]

vue). Le calcul de la table *summary delta* est le calcul de la table prepare_modifications selon la définition de la vue : les insertions et suppressions (donc les mises à jour) sont groupées. L'aspect incrémentiel de l'algorithme réside dans l'ajout du ± 1 pour l'attribut *cnt* et de $\pm$*population* pour la somme de *population*. Par exemple, si l'on ajoute, en 2008, dans la province de Québec, parmi les hommes ayant le statut de citoyen canadien, 38 naissances, et que 30 hommes de 44 ans et 49 hommes de 25 ans deviennent nouveaux citoyens, la table delta pour la première vue contiendra une mesure de population (2008, homme, citoyen, Québec) de $38 + 30 + 49 = 117$, et l'attribut *cnt*[7] vaudra $1 + 1 + 1 = 3$.

```
CREATE MATERIALIZED VIEW Can_population_annee_sexe_statut_province
AS SELECT annee.d_annee_key, sexe.d_sexe_key, statut.d_statut_pop_key,
province.province_sk, SUM(f.population) population, SUM(count(*)) as cnt_all
FROM Province_fact f, Annee annee, Sexe sexe, Statut statut, Province province
WHERE f.d_annee_key = annee.d_annee_key
AND f.d_sexe_key = sexe. d_sexe_key
AND f.d_statut_pop_key = statut.d_statut_pop_key
AND f.PROVINCE_CODE = province.SK
GROUP BY annee.d_annee_key, sexe.d_sexe_key, statut.d_statut_pop_key, province.
    province_sk;

CREATE MATERIALIZED VIEW Can_population_annee_sexe
AS SELECT f.d_annee_key, f.d_sexe_key, SUM(f.population) population
FROM Can_population_annee_sexe_statut_province f
GROUP BY f.d_annee_key, f.d_sexe_key;

CREATE MATERIALIZED VIEW Can_population_annee
AS SELECT SUM(f.population) population,
FROM Can_population_annee_sexe f
GROUP BY f.d_annee_key;
```

FIG. 3.15 – Code SQL de création des trois exemples de vues matérialisées.

La phase de rafraîchissement (Fig. 3.17) est l'exécution d'un algorithme (procédure) qui applique les changements dûs aux mises à jour et contenus dans la table *summary delta* prepare_modifications sur la vue ; pour chaque tuple de la table delta, on regarde

[7] *cnt* recense le nombre d'insertions moins le nombre de suppressions contenues dans les mises à jour

```
CREATE MATERIALIZED VIEW prepare_modifications
AS SELECT d_annee_key, d_sexe_key, d_statut_pop_key, province_sk, SUM(
    _population) as cnt_population, SUM(_count) as cnt
FROM  ( SELECT d_annee_key, d_sexe_key, d_statut_pop_key, province_sk, +
    population as _population, 1 as _count FROM log_insertions )
UNION ALL
    (SELECT d_annee_key, d_sexe_key, d_statut_pop_key, province_sk, -population
        as _population, -1 as _count FROM log_deletions )
GROUP BY d_annee_key, d_sexe_key, d_statut_pop_key, province_sk;
```

FIG. 3.16 – Fonction de propagation de l'algorithme *Summary-delta table.*

s'il existe déjà dans la vue, si oui : on ajoute (pour la somme) sa valeur à la valeur existante et on met à jour le compteur, et sinon on insère le nouveau tuple. En reprenant notre exemple, si le tuple (2008, homme, citoyen, Québec) existe déjà (ce qui est très probable), on ajoute la valeur 117 à la mesure existante. Sinon on crée ce nouveau tuple avec une valeur de 117 hommes.

L'algorithme du *Summary Delta Table* fonctionne avec les agrégations qui se maintiennent seules (*self-maintenable aggregates*). Une fonction d'agrégation est dite ***self-maintainable*** si les nouvelles valeurs peuvent être calculées uniquement à partir des anciennes valeurs et des modifications sur les données de base [Mumick *et al.*, 1997]. Les fonctions MIN() et MAX() sont *self-maintainable* par rapport à l'insertion, mais pas la suppression ; AVG() est *self-maintainable* pour les deux si on ajoute un attribut de comptage des tuples (count) ; SUM(), COUNT() sont *self-maintainable* par rapport à l'insertion et la suppression.

Pour maintenir une lattice de vues (*Nested Materialized Views* à gauche en Figure 3.18) le principe est de généraliser la méthode *summary-delta* à plusieurs vues. Si on considère le cube sous forme de lattice combinée, les vues dépendantes sont dans cette lattice. Il suffit de calculer la première table delta et de propager les modifications aux autres. La mise à jour des vues se fait donc en trois étapes :

1. calcul de la table *summary delta* de base
2. propagation des changements aux autres tables *summary delta*
3. application de toutes les tables *summary delta* sur leur vues respectives.

```
Pour chaque tuple dt dans prepare_modifications:
 On nomme t=
  (SELECT * FROM Can_population_annee_sexe_statut_province f
  WHERE f.d_annee_key =dt.d_annee_key AND f.d_sexe_key=dt.d_sexe_key
  AND f.d_statut_pop_key=dt.d_statut_pop_key AND f.province_sk=dt.province_sk)

  Si t n'existe pas,
   insérer le tuple dt dans Can_population_annee_sexe_statut_province
  Sinon /*si t existe*/
   si dt.cnt+t.count=0,
     effacer tuple t de Can_population_annee_sexe_statut_province
   sinon
     update tuple t.count+=dt.cnt,
     t.population+=dt.population
```

FIG. 3.17 – Fonction de rafraîchissement de l'algorithme *Summary-delta table.*

Comme le montre la Figure 3.18 avec les exemples définis en Figure 3.14, on a deux lattices : une pour les vues et une pour les tables delta. Le coût de la mise à jour C_U de toutes les vues m_j devient : $C_U = f_u \times \sum_{m_j \in M} c_u(m_j)$

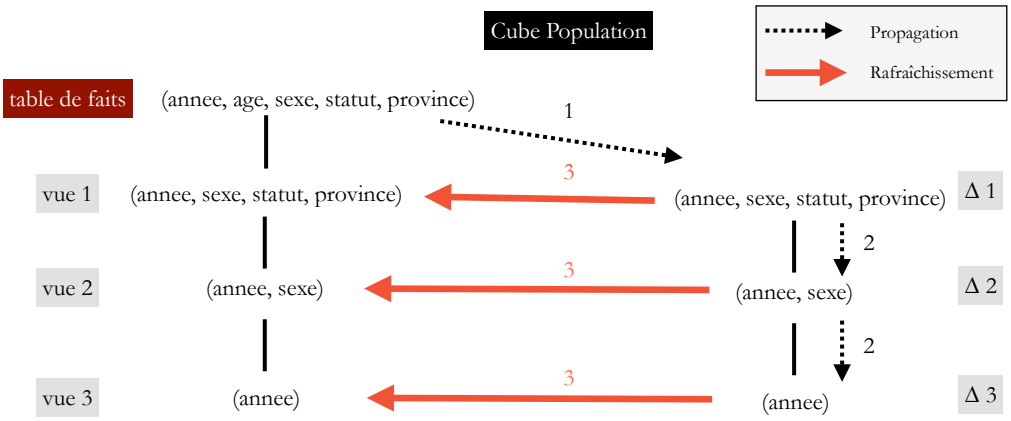


FIG. 3.18 – Propagation et rafraîchissement pour une lattice de vues.

Les vues matérialisées dans Oracle 10g

Oracle propose des fonctionnalités pour rafraîchir les vues en concordance avec les stratégies énoncés en 2.2.5 [Oracle, 2002]. Parmi elles, on compte différents modes d'exécution :

* *on commit* : maintenance immédiate
* *on demand* : maintenance différée à la demande (manuelle)
* *scheduled* : maintenance différée périodique (automatique)

et plusieurs méthodes de rafraîchissement :

* *complete refresh* : recalcule entièrement la vue matérialisée
* *fast refresh* : rafraîchit incrémentiellement la vue matérialisée, à l'aide du log des modifications sur les données de base
* *fast PCT refresh* : rafraîchit la vue en recalculant uniquement les partitions affectées par les mises à jour
* *forced refresh* : essaye le fast refresh et si ce n'est pas possible fait un complete refresh
* *never* : jamais rafraîchie

Le rafraîchissement des agrégations est l'étape la plus coûteuse de la mise à jour. Les grands fournisseurs de systèmes décisionnels avaient besoin de réduire le temps de recalcul des agrégations, c'est pour cette raison que de nombreux travaux se sont concentrés sur la maintenance incrémentielle des vues matérialisées, puis les solutions trouvées, les travaux se sont orientés vers les dimensions. Le domaine des entrepôts SOLAP est beaucoup plus jeune, quid des avancées sur les techniques de mise à jour des cubes spatiaux ?

3.4 Des méthodes à inventer pour les cubes spatiaux

Le fait de manipuler des données spatiales ajoute de la complexité au cube de données, et donc également à son processus de mise à jour. **La propagation** des mises à jour au sein du cube ne se fait plus uniquement entre les dimensions et la table de faits détaillés, et entre les faits détaillés et les faits agrégés, mais aussi au sein de la dimension spatiale géométrique, comme l'illustre la Figure 3.19. Si seul le niveau le plus détaillé est

fourni, il faut mettre à jour les niveaux supérieurs de la hiérarchie spatiale à l'aide des mises à jour sur le niveau détaillé. Cette mise à jour nécessite des opérateurs d'agrégation spatiaux tels que l'*union*, etc. Il existe malheureusement aujourd'hui très peu d'outils ETL capables d'intégrer des fonctionnalités propres aux entrepôts de données et des composantes spatiales (cf. 2.3.2), et les opérateurs d'agrégation spatiale sont à ce jour peu connus [Grenier, 2007]. Nous verrons en 4.2.2 qu'il est possible d'établir un parallèle entre la hiérarchie de la dimension spatiale et les agrégations. Les niveaux supérieurs sont déduits des niveaux inférieurs. Nous envisageons donc uniquement le cas où seul le niveau détaillé est fourni.

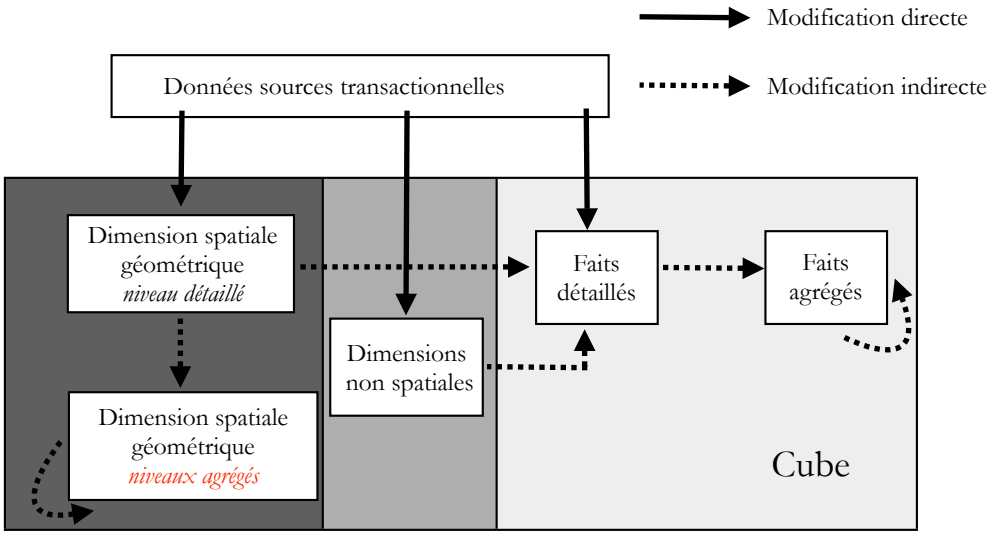


FIG. 3.19 – Processus de propagation des mises à jour dans un cube spatial.

Les stratégies existants pour la mise à jour des cubes spatiaux sont détaillées dans le Tableau 3.2. On dénote un manque de méthodes et de solutions pour les mises à jour incrémentielles et pseudo-incrémentielles. Actuellement la mise à jour des cubes spatiaux est souvent effectuée manuellement et avec plusieurs logiciels : des outils ETL pour la mise à jour des données multidimensionnelles et des logiciels tels que FME ou ArcGIS pour la mise à jour des données spatiales.

Un cube spatial peut contenir trois types de mesures :

1. mesures numériques.

2. mesures spatiales numériques

3. mesures spatiales géométriques

Les agrégats des mesures numériques peuvent être rafraîchis à l'aide des techniques définies en 3.3 et des méthodes ont été définies pour le rafraîchissement des agrégats spatiaux numériques [Papadias *et al.*, 2001]. Le calcul à la volée d'agrégats spatiaux est complexe et coûteux en temps de calcul, il est donc nécessaire de les précalculer et de les matérialiser afin d'obtenir un temps de réponse rapide [Pedersen et Tryfona, 2001]. Ces agrégats doivent ensuite être rafraîchis lors de la mise à jour de cube, malheureusement aucune méthode n'existe pour le rafraîchissement des agrégats géométriques.

Ce manque de méthodes est dû au fait que la dimension spatiale dans les cubes de données est principalement utilisée à des fins de visualisation et non d'analyse. Aucune recherche à notre connaissance, ne présente des cubes contenant des mesures spatiales géométriques et certains travaux commencent à s'intéresser aux mesures spatiales numériques [Papadias *et al.*, 2001; Pedersen et Tryfona, 2001; Malinowski et Zimányi, 2004; Raffaetà *et al.*, 2007; Kuijpers et Vaisman, 2007].

Question	Stratégie		Existence
Comment ?	**Reconstruction totale**		Courant.
	Incrémentielle		N'existe pas pour les agrégations spatiales et les dimensions spatiales.
	Pseudo-incrémentielle		N'existe pas pour les agrégations spatiales et les dimensions spatiales.
	Partitionnement		Existe [Lambert, 2006].
Quand ?	**Immédiate**		N'existe pas.
	Différée	*paresseuse (lazy)*	N'existe pas.
	Différée	*délai forcé*	N'existe pas.
	Différée	*pré-établie*	N'existe pas.
	Différée	*à la demande*	Exécution manuelle de procédures.

TAB. 3.2 – Etat de l'art des stratégies de mise à jour de cube spatial.

3.5 Conclusion du chapitre

Le chapitre 3 a expliqué en détail les méthodes existantes pour réaliser les différentes étapes du processus de mise à jour de cube dans un contexte d'entrepôt de données non spatiales : l'extraction des mises à jour, leur intégration dans le cube et le rafraîchissement des agrégations. Plusieurs solutions existent pour l'extraction des mises à jour. Le choix de l'une ou l'autre solution se fera en fonction des besoins du projet global de gestion de l'entrepôt de données. Le chapitre 3 traite ensuite en détail de l'intégration des mises à jour dans le cube de données. Cette intégration consiste en la mise à jour des dimensions et de la table de faits. Les différentes techniques pour mettre à jour les dimensions on été détaillées. La technique de Slowly Changing Dimensions reste la plus adoptée, nous avons choisi de nous pencher davantage sur cette dernière. La dernière étape de la mise à jour de cube consiste en le rafraîchissement des agrégations. Pour aider à comprendre ce processus, nous avons explicité les différentes implantations possibles pour les agrégats dans un cube. Ensuite les techniques de vues matérialisées ont été abordées et expliquées.

Le chapitre 3 présente un état de l'art assez complet des techniques existantes pour la mise à jour incrémentielle de cubes non spatiaux. Ces techniques semblent matures et ont fait leur preuves. La dernière section aborde la mise à jour des cubes spatiaux et explique en quoi les techniques de mise à jour incrémentielle des cubes non spatiaux sont insuffisantes pour mettre à jour incrémentiellement des cubes spatiaux. Les raisons principales sont que les dimensions spatiales contiennent des niveaux agrégés dont il faut mettre à jour la géométrie, et que les agrégations ne sont plus uniquement sur des mesures numériques, mais également sur des mesures spatiales numériques et géométriques, nécessitent des traitements spécifiques. Il conclut sur la nécessité de définir des méthodes incrémentielles pour les cubes spatiaux : les entrepôts de données spatiales sont destinés à exploiter la puissance des analyses spatiales et pour cela, il est primordial d'inventer des méthodes et processus pour la mise à jour des dimensions et des mesures spatiales. Un outil ETL capable de manipuler des données multidimensionnelles spatiales (c.-à-d. combinant fonctions spatiales et fonctions ETL multidimensionnelles classiques) serait

également un atout important.

Chapitre 4

Proposition de méthodes et processus pour une mise à jour de cube spatial incrémentielle et conception du service web

Le chapitre 3 a présenté les solutions existantes pour mettre à jour des cubes de données non spatiales. Dans le chapitre 4, nous nous inspirons des méthodes incrémentielles présentées au chapitre 3 pour suggérer de nouvelles méthodes destinées à la mise à jour incrémentielle des cubes de données spatiales. Les cubes spatiaux ont des spécificités propres qu'il faut prendre en compte (section 4.1) pour proposer des méthodes de mise à jour (section 4.2). Afin de rendre le processus interopérable et accessible à distance, nous proposerons de l'intégrer dans une architecture orientée services. La dernière section (section 4.3) détaille la conception du service web de mise à jour.

4.1 Prise en considération des spécificités des cubes spatiaux

Un cube de données spatiales contient une ou plusieurs dimensions spatiales et/ou des mesures spatiales (cf. 2.2.1). Il convient d'expliciter la modélisation conceptuelle de ces nouvelles dimensions et mesures, ainsi que leur implantation dans le cube. L'aspect spatial ajoute de nouvelles contraintes d'intégrité que nous tenterons ici d'expliquer brièvement.

4.1.1 Différentes conceptions et implantations possibles

4.1.1.1 Les types de cubes

McHugh [2008] définit deux types de cubes spatiaux :

1. les cubes spatiaux *vectoriels* ; ils contiennent des faits ou membres de dimension représentant des éléments discrets. La géométrie de ces éléments est sous forme vectorielle.
2. les cubes spatiaux *raster* ; ils contiennent des faits représentant des éléments ou phénomènes continus de l'espace. Chaque cellule de la grille raster correspond à un fait.

Les cubes *vectoriels* et les cubes *rasters* sont conceptuellement très distincts, leur mise à jour ne va pas mettre en jeu les mêmes méthodes.

4.1.1.2 Les dimensions spatiales

Bédard *et al.* [2001] distinguent trois principaux types de dimensions spatiales (Fig. 4.1) :

1. *non géométrique* : ne contient pas d'attribut géométrique (ni vectoriel, ni raster)
2. *mixte géométrique* : contient des niveaux avec géométrie vectorielle et des niveaux sans géométrie

3. *géométrique* : tous les niveaux ont un attribut géométrique vectoriel

FIG. 4.1 – Les principaux types de dimensions spatiales pour un cube

Suite aux recherches de McHugh [2008] sur le potentiel d'intégration du raster dans les cubes spatiaux, de nouveaux types de dimensions ont été définis. Le lecteur est invité à se référer à [McHugh, 2008] pour plus de précisions. Dans la suite du mémoire, sauf si précisé, le terme cube spatial se réfère à un cube spatial vectoriel. Nous définirons ainsi des méthodes de mise à jour pour des cubes spatiaux vectoriels ; la mise à jour des cubes raster est plus simple et suscite moins d'intérêt (cf. 4.2.4).

Il existe différents types de hiérarchies de dimensions dans les cubes spatiaux, nous ne les détaillerons pas toutes, car ce n'est pas l'objet de la présente recherche. Le lecteur est invité à se référer à [Malinowski et Zimányi, 2005] pour plus de détail. L'implantation de la dimension diffère selon chaque type. Nous baserons nos exemples sur le type de hiérarchie suivant : une hiérarchie stricte[1] et balancée[2] (exemple en Fig. 3.11), car ce sont les hiérarchies les plus ordonnées et donc les plus simples à comprendre et à mettre en oeuvre.

Le stockage des données géométriques

Les données géométriques peuvent être stockées sous différents formats : dans des bases de données à référence spatiale comme Oracle Spatial ou PostGIS, dans des fichiers de type CAD ou SIG (shapefiles, fichiers mapinfo, etc.), sous forme de fichiers XML

[1] Chaque membre possède un seul parent

[2] Tous les membres d'un même niveau sont à égale distance du niveau supérieur

(GML), etc. La plupart des cubes de données spatiales actuels stockent les données géométriques dans un fichier de format SIG relié au cube par le serveur SOLAP [Rivest *et al.*, 2001; Gomez *et al.*, 2007]. Les données spatiales ne sont pas intégrées dans le cube de données directement.

Pour intégrer les données spatiales dans le cube, il est nécessaire de stocker les géométries dans la base de données qui contient le cube. Il existe deux manières de le faire (Fig. 4.2) :

- **dimension spatiale étoile** avec un champ de géométrie par niveau
- **dimension spatiale étoile modifiée** avec une table de géométrie par niveau liée à la table de dimension

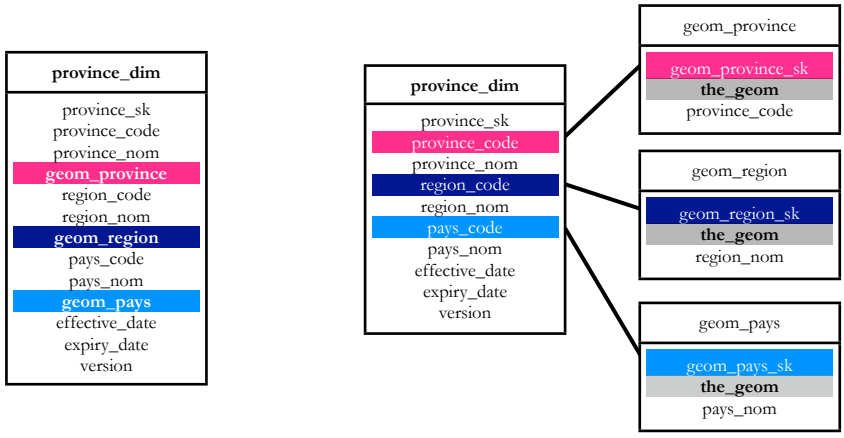


FIG. 4.2 – Dimension spatiale étoile et étoile modifiée

4.1.1.3 Les mesures spatiales

Les cubes spatiaux amènent trois nouveaux types de mesures [Bédard *et al.*, 2008]. Ils contiennent donc les quatre types de mesures suivantes :

1. *mesure non spatiale* : valeur numérique dérivée de données numériques ou textuelles (p. ex. salaire, prix , quantité, coût, etc.).
2. *mesure spatiale numérique* : valeur numérique dérivée de données spatiales (p. ex. densité, aire, nombre d'éléments dans un espace).

3. *mesure spatiale géométrique* : ensemble de coordonnées représentant un élément spatial ou pointeurs sur cet élément (p. ex. trajectoire d'une voiture, délimitation d'un incendie de forêt).

4. *mesure spatiale complète* : combinaison d'une valeur numérique avec sa géométrie associée (p. ex. nombre de kilomètres avec une trajectoire de véhicule).

Comme nous l'avons remarqué en 3.4, ces considérations sont assez théoriques et encore peu mises en pratique. Aucune recherche ne présente des mesures spatiales géométriques, ce qui rend leur mise à jour complexe à définir. La mise à jour des agrégats spatiaux géométriques se fera à l'aide d'opérateurs d'agrégation géométriques. Ces opérateurs sont nombreux, on recense par exemple les fonctions : *rectangle englobant, enveloppe convexe, centre de gravité* ; les plus élémentaires restant l'*union*, l'*intersection* et la *différence symétrique*. Leur classification fait actuellement l'objet de recherches [Grenier, 2007].

4.1.2 Établissement des contraintes d'intégrité spatiales

Une contrainte d'intégrité est un ensemble de règles de contrôle de cohérence des valeurs saisies dans un système (le plus souvent SGBD). Dans une base de données à référence spatiale, une contrainte d'intégrité spatiale est une règle permettant de définir les relations géométriques entre deux objets afin de s'assurer qu'il n'y a pas d'incohérence sémantique entre eux (p. ex. une maison ne peut être située au milieu d'une autoroute). Les contraintes d'intégrité sont présentes dans le domaine transactionnel afin de garantir la qualité des données alors que dans un cube de données, leur but est de garantir la qualité de décision [Salehi *et al.*, 2007]. Elles s'appliquent sur les dimensions et sur les mesures.

Salehi *et al.* [2007] recensent trois types de contraintes d'intégrité propres aux cubes de données :

- les contraintes *spatio-temporelles*
- les contraintes *géométriques*, au sein de la dimension spatiale
- les contraintes *non géométriques*, au sein des autres dimensions

Nous nous intéressons ici uniquement aux contraintes géométriques, car ce sont principalement elles qui vont entrer en jeu lors de la mise à jour de la dimension spatiale.
Selon Salehi *et al.* [2007], les contraintes d'intégrité spatiales géométriques s'expriment toujours entre deux niveaux d'une hiérarchie spatiale. Or les hiérarchies spatiales peuvent être à plusieurs échelles (échelles différentes suivant les niveaux) ou à simple échelle de représentation. Dans le premier cas, les contraintes doivent tenir compte de la généralisation cartographique, puisque les niveaux sont représentés à différentes échelles. Par exemple pour la dimension de la Figure 4.2, le niveau province pourrait être représenté à l'échelle 1/25000 et le niveau région à l'échelle 1/50000. Les régions auront subit une généralisation cartographique, l'union des provinces de la région *Centre* n'est donc plus strictement égale à la région *Centre*, mais approximativement (ex : 98% de recouvrement). Salehi *et al.* [2007] distinguent alors deux types de contraintes : les *contraintes dures*, pour les hiérarchies à simple échelle et les *contraintes douces* incluant un intervalle de liberté, pour les hiérarchies multiéchelles. Dans notre exemple, on pourrait poser la contrainte douce suivante : l'union des provinces de chaque région doit recouvrir au moins 98% du territoire de la région (la contrainte dure imposerait l'égalité).
Pedersen et Tryfona [2001] posent des contraintes sur les dimensions et sur les mesures spatiales : la hiérarchie spatiale doit être stricte et balancée pour garantir la sommabilité des mesures, et les mesures spatiales numériques doivent être disjointes et distributives ou algébriques sur la fonction d'agrégation.
Ferri *et al.* [2000] souhaitent faire le lien entre les bases de données statistiques (SDB) et les bases de données géographiques, pour permettre aux utilisateurs de SDB d'effectuer des analyses spatiales. Ils adaptent la contrainte *complete containment* (SDB) aux données spatiales en définissant une nouvelle contrainte appelée *full contains* au sein d'une dimension spatiale d'une base de données statistique géographique. Soient NG_1 et NG_2 deux niveaux d'une hiérarchie spatiale géométrique (Fig. 4.3), NG_1 *full contains* NG_2 si et seulement si :

- $\bigcup_i NG_{2i} = \overline{NG_1}$; l'union des membres de NG_2 est égale à la fermeture topologique de NG_1 (selon l'exemple de la Fig. 4.2 l'union des provinces est égale à

l'ensemble fermé des régions).

- $\bigcap_i NG_{2i} = \emptyset$; les membres de NG_2 sont tous disjoints (les provinces de la Fig. 4.2 sont disjointes).
- $\bigcap_i NG_{1i} = \emptyset$; les membres de NG_1 sont tous disjoints (les régions de la Fig. 4.2 sont disjointes).

La relation *full contains* garantit la sommabilité des mesures selon la hiérarchie spatiale.

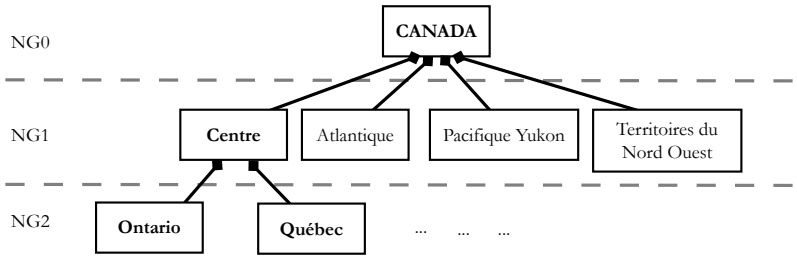


FIG. 4.3 – Dimension spatiale *Provinces*.

Suite à ces considérations, nous baserons nos recherches en adoptant les contraintes d'intégrité suivantes :

* **Les hiérarchies spatiales géométriques devront respecter la contrainte *full contains***. Dans l'exemple de la Figure 4.3, si cette contrainte est respectée, la région *Centre* est l'union des deux provinces *Ontario* et *Québec*. Si la contrainte n'était pas respectée, alors la région *Centre* serait peut-être plus grande que l'union de l'*Ontario* et du *Québec*, sa mise à jour devient plus compliquée, car il est nécessaire d'identifier la zone sans provinces, et de connaître ses mises à jour.
* **Les contraintes spatiales seront dures et nos hiérarchies à simple échelle**, car nous voulons nous affranchir de la généralisation cartographique.
* **Toutes nos hiérarchies seront strictes et balancées.** Nous n'aborderons pas le cas de la mise à jour pour des hiérarchies plus complexes.
* **Les mesures spatiales numériques devront être distributives sur l'union** (détail en 4.2.3).
* **Nous ne considérons pas de superposition entre les membres d'un même niveau spatial.** Dans ce cas, les membres d'un même niveau ne s'intersectent

pas. S'ils s'intersectent, c'est qu'une mise à jour est survenue et il y a une décision à prendre pour propager ces mises à jour aux membres frères intersectés. Cette dernière contrainte vient complexifier le processus, mais nous permet de distinguer deux cas abordés en 5.3.2, le cas où l'ETL fait confiance dans l'intégrité ses données sources et le cas contraire.

Ces contraintes nous permettent de proposer des méthodes de base en travaillant sur des cas simplifiés[3].

4.2 Solutions proposées pour la mise à jour de cube spatial

Les cubes spatiaux présentent de nouvelles caractéristiques. Les méthodes classiques de mise à jour (pour les cubes non spatiaux) doivent être revues ou adaptées pour supporter la mise à jour les cubes spatiaux. Cette section présente une typologie des modifications possibles au sein d'une dimension spatiale ainsi que des méthodes incrémentielles pour propager ces modifications au sein de la dimension spatiale. Pour finir, les méthodes pour la mise à jour des mesures spatiales seront explicitées.

4.2.1 Typologie des mises à jour au sein de la dimension spatiale

Les données contenues dans la dimension spatiale sont des données spatio-temporelles, c.-à-d. des données spatiales qui évoluent dans le temps. Les membres de la dimension spatiale représentent des objets spatiaux dont la **position** ou la **forme** peuvent changer au cours du temps. Ces changements peuvent être plus ou moins fréquents selon les objets (p. ex. les masses d'air bougent continûment alors que les frontières des pays sont assez figées).

D'après les travaux de [Badard, 2000; Pouliot *et al.*, 2004; Bédard *et al.*, 2003;

[3]Le temps imparti pour une maîtrise ne permettant pas de traiter exhaustivement tous les cas.

Rageul, 2007], nous recensons les modifications directes susceptibles d'avoir lieu sur les membres de la dimension spatiale :

- **modification de la géométrie d'un membre**. Par exemple, suite à de nouvelles mesures GPS, on pourrait mettre à jour la géométrie de la province du Québec.
- **ajout d'un membre**. L'ajout d'un membre dans la dimension spatiale géométrique s'accompagne de l'ajout d'une géométrie pour ce membre. Dans le cas de la dimension « Territoires », contenant les provinces, on pourrait ajouter une nouvelle province au Canada : le Vermont (antérieurement État des États-Unis).
- **suppression d'un membre**. On pourrait vouloir supprimer la province de l'Ile du Prince Édouard de notre dimension spatiale (virtuellement).
- **reclassification d'un membre dans la hiérarchie**. Par exemple : la province du Québec devient un pays, et se situe maintenant au même niveau hiérarchique que le Canada.
- **fusion de n membres d'un même niveau en un membre**. Par exemple : la fusion des provinces du Québec et de l'Ontario pour donner une seule province.
- **scission d'un membre en n sur un même niveau**. Par exemple : la scission de l'Ontario en trois provinces.

4.2.2 La propagation incrémentielle des mises à jour au sein des dimensions spatiales

Plusieurs scénarios sont possibles pour la livraison des données servant à construire la dimension spatiale (livraison des sources de données au cube) :

1. les données géométriques de tous les niveaux sont tous fournies par les systèmes sources. Les systèmes sources peuvent être des bases de données transactionnelles, des fichiers de données ou encore tout autre type de source de données alimentant le cube.
2. seules les données géométriques relatives au niveau le plus granulaire de la hiérarchie spatiale sont fournies (dans notre exemple de la Figure 4.3, le niveau le

plus granulaire est le niveau provinces), et dans ce cas il faut générer les géométries des niveaux supérieurs (région et pays).

Notre recherche se positionne dans le second cas : nous considérons que seules les géométries relatives au niveau le plus granulaire de la hiérarchie spatiale sont fournies au cube par les sources, et que l'ETL doit générer et mettre à jour les géométries des niveaux supérieurs.

Nous avons étudié précédemment le processus de propagation des mises à jour au sein de la lattice du cube (cf 3.3.3.3). La hiérarchie spatiale est également organisée en lattice ; il est possible d'établir une analogie entre des deux lattices (Tab. 4.1). Tout comme on matérialise les agrégations, on matérialise les niveaux supérieurs de la hiérarchie spatiale. L'analogie s'étend : le niveau détaillé est comparable aux faits détaillés, les niveaux supérieurs aux faits agrégés et la géométrie d'un membre est analogue à une mesure non spatiale.

Les niveaux supérieurs de la hiérarchie spatiale sont donc comparables aux agrégations : ils sont alors conçus et créés par l'équipe de l'entrepôt de données à des fins d'analyse, et doivent rester le plus indépendants des sources, c'est pour cette raison que nous nous positionnons dans le second cas.

Analogie	
géométrie d'un membre du niveau détaillé	mesure d'un fait détaillé
géométrie d'un membre de niveau supérieur	mesure d'un fait agrégé
niveau détaillé	table de faits détaillés
niveau supérieur	table de faits agrégrés (vue)
lattice de la hiérarchie spatiale	lattice du cube

TAB. 4.1 – Analogie entre le lattice de la dimension spatiale et la lattice du cube

La propagation des mises à jour au sein de la dimension spatiale sera donc traitée par analogie avec la propagation des mises à jour au sein de la lattice des vues, à la différence que les modifications directes ne sont plus uniquement des ajouts (cf. typologie donnée en 4.2.1). Nous optons pour une propagation **incrémentielle** des mises à jour au sein de la hiérarchie spatiale, par analogie avec la propagation incrémentielle des mises à jour

vue dans le Tableau 2.2. Au sein de la lattice de cube, on met à jour les agrégations ; ici, au sein de la lattice de la hiérarchie spatiale, on met à jour les géométries des niveaux supérieurs.

Nous considérons que la dimension spatiale est connue à priori, ce qui nous évite d'avoir à la calculer comme le font Stefanovic *et al.* [2000] et Papadias *et al.* [2001] à l'aide d'algorithmes spéciaux et de techniques d'indexation par rectangle englobant.

Nous résumons dans le Tableau 4.2 les **impacts des modifications directes** énoncées en 4.2.1. Les contraintes d'intégrité jouent un rôle essentiel dans l'étape de propagation. Pour expliquer ce tableau, prenons l'exemple de la dimension spatiale représentée en Figure 4.3 :

- Suite à une modification de la géométrie du *Québec* il faut vérifier si le territoire couvert par le *Québec* n'intersecte pas celui couvert par l'*Ontario*, et si l'union de l'*Ontario* et du *Québec* est toujours égale à la région *Centre*. Si la première condition est respectée, mais pas la seconde, il faudra recalculer la géométrie de la région *Centre* et propager les modifications aux niveaux supérieurs, c.-à-d. refaire les mêmes vérifications en prenant pour donnée de départ la région *Centre*. Le processus est récursif.
- Suite à un ajout d'une nouvelle province *Vermont* dans la région *Centre*, il faut vérifier si le territoire couvert par le *Vermont* n'intersecte pas celui couvert par le *Québec* et l'*Ontario*. Ensuite il faudra recalculer la géométrie du *Centre* et propager les modifications aux niveaux supérieurs.
- Suite à la suppression de la région du *Centre* (suppression totale de la région et de ses provinces), il faut supprimer les provinces *Québec* et *Ontario* et recalculer la géométrie du pays *Canada*.
- Si le *Québec* passe dans la région *Atlantique*, il faut recalculer la géométrie de l'ancienne région à laquelle appartenait le *Québec* : le *Centre* et recalculer la géométrie de la nouvelle région *Atlantique*.
- Si le *Québec* et l'*Ontario* fusionnent, aucune modification ne doit être apportée à la région *Centre* et donc aucune non plus au pays *Canada*.
- Si le *Québec* se divise en deux provinces, les géométries de ces deux provinces

Modification directe	Vérification des contraintes	Effets à propager
modification de la géométrie d'un membre détaillé	par rapport au père et aux frères[a]	– contraintes non validées par rapport au père : mise à jour de la géométrie du père et des ancêtres – contraintes non validées par rapport aux frères : autre décision à prendre[b]
ajout d'un membre	par rapport au père et aux frères	– mise à jour de la géométrie du père et des ancêtres – contraintes non validées par rapport aux frères : autre décision à prendre
suppression d'un membre	aucune	– supprimer les fils – recalcul de la géométrie du père et des ancêtres
reclassification d'un membre	par rapport aux fils, aux frères et au père	– mise à jour de la géométrie de l'ancien père et des anciens ancêtres – mise à jour de la géométrie du nouveau père et des nouveaux ancêtres
fusion de n membres d'un même niveau en un membre	aucune	aucun
scission d'un membre en n sur un même niveau	aucune	aucun

[a] Les frères sont les membres appartenant au même niveau.

[b] Par exemple si la géométrie mise à jour du Québec nous est fournie mais qu'elle intersecte la géométrie de l'Ontario. Doit-on modifier ou non la géométrie de l'Ontario ? Il peut s'agir d'une erreur de la part du système source, il y a une décision à prendre ici.

TAB. 4.2 – Propagation incrémentielle des mises à jour au sein de la dimension spatiale.

seront fournies par les systèmes sources et donc leur union sera égale à la géométrie du *Québec*, aucune modification n'est nécessaire aux niveaux supérieurs.

Comme le montre le tableau, la validation des contraintes est surtout présente dans le premier cas : la modification de la géométrie d'un membre.

4.2.3 La propagation des mises à jour pour les mesures spatiales

Outre la mise à jour des dimensions, la mise à jour de cube implique également la mise à jour des mesures du cube. La mise à jour des mesures s'effectue en deux étapes : la saisie des modifications directes dans la table de faits détaillés et le rafraîchissement des agrégations (Fig. 3.19). Nous avons vu précédemment les méthodes et techniques existantes pour mettre à jour des mesures non spatiales (cf. 3.3). Nous nous intéressons ici aux mesures spatiales numériques et géométriques.

La mise à jour des mesures spatiales numériques

Dans le contexte des mesures spatiales numériques, les systèmes sources vont garder les mesures détaillées alors que la table de faits détaillés du cube contiendra souvent les mesures spatiales numériques déjà agrégées [Papadias *et al.*, 2002]. Deux principales raisons viennent appuyer ce propos : d'une part pour des raisons légales, certaines données détaillées ne doivent pas être dévoilées (p. ex. la position des usagers de téléphones mobiles) et d'autre part les données détaillées sont souvent insignifiantes (prises seules), mais agrégées, elles apportent de l'information (p. ex. il est inutile de savoir qu'un véhicule circule sur le viaduc de Millau à 10 h, mais il est plus utile de connaître le nombre de véhicules qui ont traversé le viaduc entre 9 h et 10 h). Le peuplement de la table de faits détaillés du cube nécessite une pré-agrégation des mesures.

Il est nécessaire de définir de nouvelles méthodes pour l'intégration des mesures dans la table de faits détaillés, le rafraîchissement des agrégations ne concerne que des mesures numériques et s'effectue à l'aide des mêmes méthodes que pour les mesures non spatiales (Fig. 4.4). Le traitement spatial des données intervient en effet entre l'acquisition des données source et leur intégration aux faits.

Imaginons une table de faits détaillés mesurant le nombre de voitures par segment de route et par heure, comme illustré en Figure 4.5. Pour la peupler, il faut déterminer sur quel segment circule chaque voiture (point jaune) à l'aide de sa position GPS et des coordonnées spatiales des segments de route. Ceci étant réalisé, à chaque couple

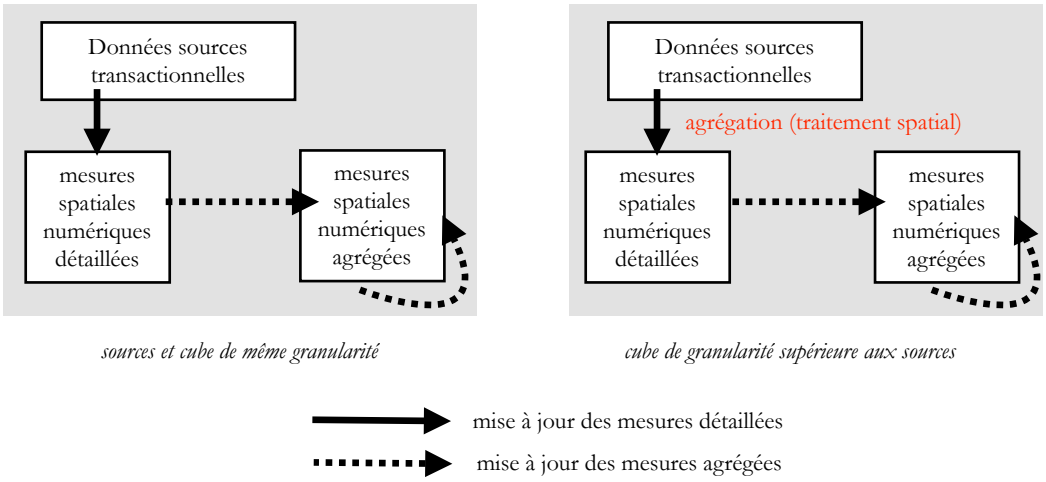


FIG. 4.4 – Mise à jour des mesures spatiales numériques pour différentes granularités.

d'identifiants *segment_id, heure_id* est attribué une mesure du nombre de voitures.

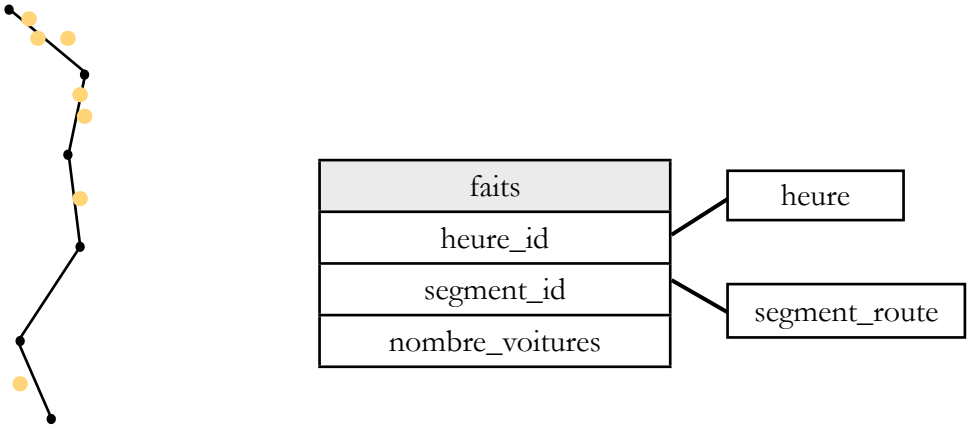


FIG. 4.5 – Exemple de mesure spatiale numérique : nombre de voitures par segment de route et par heure.

La propagation des effets des mises à jour aux agrégations s'effectue entre la table de faits détaillés et les agrégats, et comme nous pouvons le remarquer dans cet exemple, elle est similaire à la propagation pour des mesures numériques non spatiales puisqu'il n'y a aucun traitement spatial entre les faits détaillés et les agrégats.

On recense peu de travaux dans le domaine de la mise à jour des mesures spatiales numériques, parmi eux :

* Pedersen et Tryfona [2001] proposent des contraintes pour la sommabilité des

mesures spatiales numériques :

- la fonction d'agrégation numérique doit être distributive sur l'*union*.

  Soient $f()$ une fonction d'agrégation, $g()$ une mesure spatiale, A et B deux membres de la dimension spatiale géométrique : $g(A \cup B) = f(g(A), g(B))$.

  Par exemple, on prend $f()$ la fonction *somme* et $g()$ la mesure d'*aire*, on a bien : $aire(A \cup B) = aire(A) + aire(B)$[4].

* La plus grande contribution est celle de Papadias *et al.* [2001] qui ont défini **une méthode pour l'intégration des mesures dans la table de faits détaillés**. Papadias *et al.* [2001] ont adapté la phase de propagation de l'algorithme du *Summary-delta table* de Mumick *et al.* [1997] expliqué en 3.3.3.3, pour mettre à jour des mesures spatiales numériques de la table de faits. Le rafraîchissement des agrégations s'effectue avec la fonction de rafraîchissement du *Summary-delta*. La méthode de propagation dépend de la nature des mesures. Papadias *et al.* [2001] se placent dans le cas de la supervision de trafic avec la mesure du nombre de voitures par segment de route (cf. Fig. 4.5) Les systèmes sources fournissent un journal des positions GPS des véhicules. Il existe deux manières de peupler la table de faits :

  - incrémentielle individuelle : pour chaque tuple dans le journal, on regarde à quel segment de route il appartient et on ajoute 1 au nombre de voitures du segment, on regarde à quel segment il appartenait avant et on soustrait 1.
  - incrémentielle par lots : même procédure avec une jointure sur les segments de route.

La mise à jour des mesures spatiales géométriques

Les mesures spatiales géométriques, encore très peu implantées dans les cubes, sont plus difficiles à mettre à jour en raison de la complexité des données géométriques (traitements plus longs, gourmandes en espace mémoire, présence d'informations implicites, etc.). Il n'existe pas, à notre connaissance, de travaux proposant des méthodes pour leur

[4]Puisque nous avons établi dans nos contraintes que les membres d'un même niveau la dimension spatiale sont disjoints.

mise à jour. Nous proposons alors un nouvel algorithme *Spatial Summary-delta* pour la mise à jour des mesures spatiales, adapté de l'algorithme du *Summary-delta* (3.3.3.3). Il est aisé de faire l'analogie entre le numérique et le géométrique en comparant la *somme* avec l'*union* et la *différence* avec la *différence symétrique*. Aujourd'hui l'agrégation des mesures spatiales est souvent effectuée avec l'opérateur *union* [Stefanovic *et al.*, 2000]. Nous allons adapter l'algorithme du *Summary-delta* dans le cas de la fonction *somme*. Puisque l'algorithme du *Summary-delta* est l'algorithme utilisé pour mettre à jour les mesures numériques spatiales (cf. paragraphe précédent) et non spatiales, il nous a paru judicieux de l'adapter aux mesures spatiales géométriques. Pour illustrer le fonctionnement de L'algorithme, prenons l'exemple du cube *inondation* de la Figure 4.6.

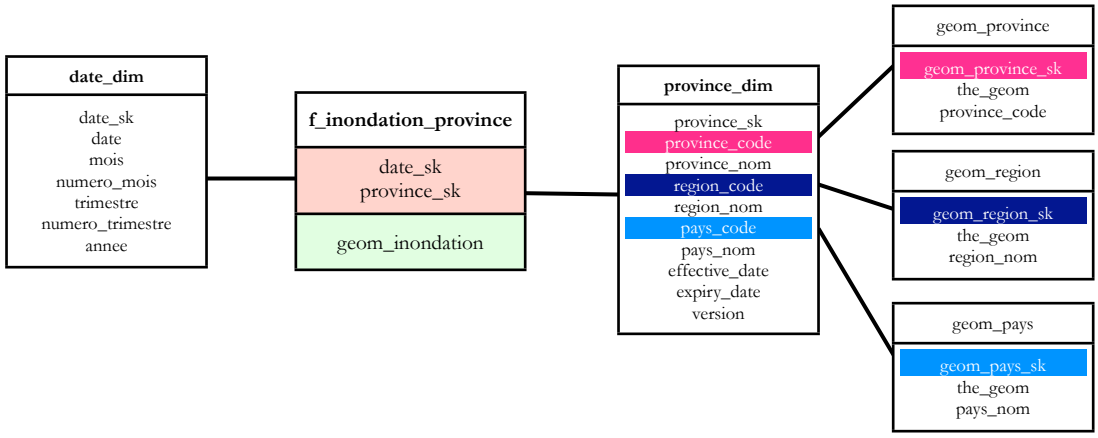


FIG. 4.6 – Cube spatial inondation.

Ce cube comprend une dimension spatiale *province* et une dimension temporelle. Sa table de faits contient une mesure spatiale géométrique polygonale représentant une surface inondée. Le cube possède deux tables d'agrégations : *inondation_province_mois* et *inondation_province_annee* (Fig. 4.7) représentant respectivement les zones inondées agrégées par {mois,année}, et par année. L'algorithme *Spatial Summary-delta* va servir à propager les mises à jour au sein de la lattice composée des trois tables suivantes : *inondation_province_mois* et *inondation_province_annee* à partir des mises à jour sur la table de faits *f_inondation_province*.

L'algorithme est toujours divisé en deux parties :

```
CREATE TABLE inondation_province_mois
AS SELECT date.annee, date.mois, province.province_code,
      geomunion(f.geom_inondation) geom_inondation
FROM f_inondation_province f, date_dim d, province_dim p
WHERE f.date_sk = d.date_sk
AND f.province_sk = p.province_sk
GROUP BY date.annee, date.mois, province.province_code;

CREATE TABLE inondation_province_annee
AS SELECT date.annee, province.province_code, geomunion(f.geom_inondation)
    geom_inondation
FROM f_inondation_province f, date_dim d, province_dim p
WHERE f.date_sk = d.date_sk
AND f.province_sk = p.province_sk
GROUP BY date.annee, province.province_code;
```

FIG. 4.7 – Exemples de tables agrégées avec mesures spatiales géométriques.

1. **propagation** : en dehors de la fenêtre de mise à jour. Crée les tables *spatial summary delta* (les tables de faits agrégés sont encore disponibles).
2. **rafraîchissement** : applique les changements inclus dans les tables *spatial summary delta* sur la table de faits agrégés concernée (les tables de faits agrégés ne sont plus accessibles).

La phase de propagation (Fig. 4.8) consiste à préparer les changements en calculant les tables *spatial summary delta*. Les tables *spatial summary delta prepare_union* et *prepare_difference* ont la même structure que la table de faits agrégés *inondation_province_mois* : les géométries insérées sont regroupées par l'opération d'*union* en un élément *geom_inondation* pour la table *prepare_union*, de même les géométries effacées sont regroupées par l'opération d'*union* en un élément *geom_inondation* de la table *prepare_difference*.

La phase de rafraîchissement (Fig. 4.9) est l'exécution d'un algorithme (procédure) qui applique les changements des table *spatial summary delta prepare_union* et *prepare_difference* sur la table agrégée ; pour chaque tuple de la table *prepare_union*, on regarde s'il existe déjà dans la table agrégée, si oui : on effectue l'***union*** de la géométrie *geom_inondation* de ce tuple avec celle du tuple présent dans la table agrégée, et sinon

```
CREATE TABLE prepare_union
AS SELECT date.annee, date.mois, province.province_code,
      geomunion(geom_inondation) geom_inondation
FROM  ( SELECT date.annee, date.mois, province.province_code, geom_inondation
    FROM log_insertions
    WHERE date.date_sk=log_insertions.date_sk
    AND province.province_sk=log_insertions.province_sk)
GROUP BY date.annee, date.mois, province.province_code;

CREATE TABLE prepare_difference
AS SELECT date.annee, date.mois, province.province_code, geomunion(
    geom_inondation) geom_inondation
FROM  ( SELECT date.annee, date.mois, province.province_code, geom_inondation
    FROM log_deletions
    WHERE date.date_sk=log_insertions.date_sk
    AND province.province_sk=log_insertions.province_sk)
GROUP BY date.annee, date.mois, province.province_code;
```

FIG. 4.8 – Fonction de propagation de l'algorithme *Spatial Summary-delta.*

on insère le nouveau tuple. De même, pour chaque tuple de la table *prepare_difference*, on regarde s'il existe déjà dans la table agrégée, si oui : on effectue la ***différence symétrique*** entre la géométrie *geom_inondation* du tuple correspondant présent dans la table agrégée avec celle du tuple, et sinon on ne fait rien. De la même manière qu'avec l'algorithme *Summary-delta*, il est possible de propager les mises à jour au sein d'une lattice de tables de faits agrégés (cf. 3.3.3.3).

Le diagramme d'activité de l'algorithme Spatial Summary-delta est fourni en Figure 4.10, il explicite le déroulement de deux étapes : propagation et rafraîchissement. Nous avons inclus les suppressions de géométries dans l'algorithme pour respecter l'analogie. Toutefois, nous ne considérons pas les suppressions dans notre typologie de *mise à jour de cube* (disponible en 2.2.3.2).

Étant donné l'envergure du développement d'un tel algorithme, il fut jugé que cela débordait le cadre du présent mémoire. De plus de telles mesures sont rares dans les cubes de données. Nous n'avons donc pas implanté l'algorithme *Spatial Summary-delta* et n'avons pas effectué de tests permettant de le valider. Nous pensons que l'algorithme proposé fonctionnerait dans le cas simple de l'*union* (resp. *différence symétrique*).

```
Pour chaque tuple dt dans prepare_union:
 On nomme t=
  (SELECT * FROM inondation_province_mois f
  WHERE dt.mois=f.mois
    AND dt. province_code =f. province_code)

 Si t n'existe pas,
  insérer le tuple dt dans inondation_province_mois
 Sinon /*si t existe*/
    t.geom_inondation=geomunion(t.geom_inondation, dt.geom_inondation)

Pour chaque tuple dt dans prepare_difference:
 On nomme t=
  (SELECT * FROM inondation_province_mois f
  WHERE dt.mois=f.mois
    AND dt. province_code =f. province_code)

 Sinon /*si t existe*/
    t.geom_inondation=difference(t.geom_inondation, dt.geom_inondation)
```

FIG. 4.9 – Fonction de rafraîchissement de l'algorithme *Spatial Summary-delta.*

Pour approfondir le sujet, il conviendrait de déterminer et de classifier les fonctions d'agrégation géométrique et de définir des algorithmes pour ces fonctions.

4.2.4 Remarque sur la mise à jour des cubes vecteurs et rasters

Les cubes *raster* modélisent des phénomènes spatiaux continus (température, humidité de l'air, répartition de la neige au sol, etc.) appelés à évoluer fréquemment contrairement aux phénomènes discrets dans les cubes *vectoriels* (bâtiments, routes, etc.), assez figés dans le temps. Il est alors opportun d'établir un parallèle entre les techniques de *Slowly Changing Dimensions (SCD), Rapidly Changing Dimensions (RCD)* (cf. 3.2) et la mise à jour des dimensions spatiales des cubes vecteurs et rasters. Les dimensions des cubes vecteurs seront mises à jour suivant la technique *SCD*, les dimensions spatiales des cubes rasters selon la technique *RCD* ; c'est ce qu'a fait McHugh [2008] en divisant la dimension spatiale raster en deux : une minidimension fixe représentant la grille raster et une autre dimension *SCD* contenant les informations sur les mesures spatiales.

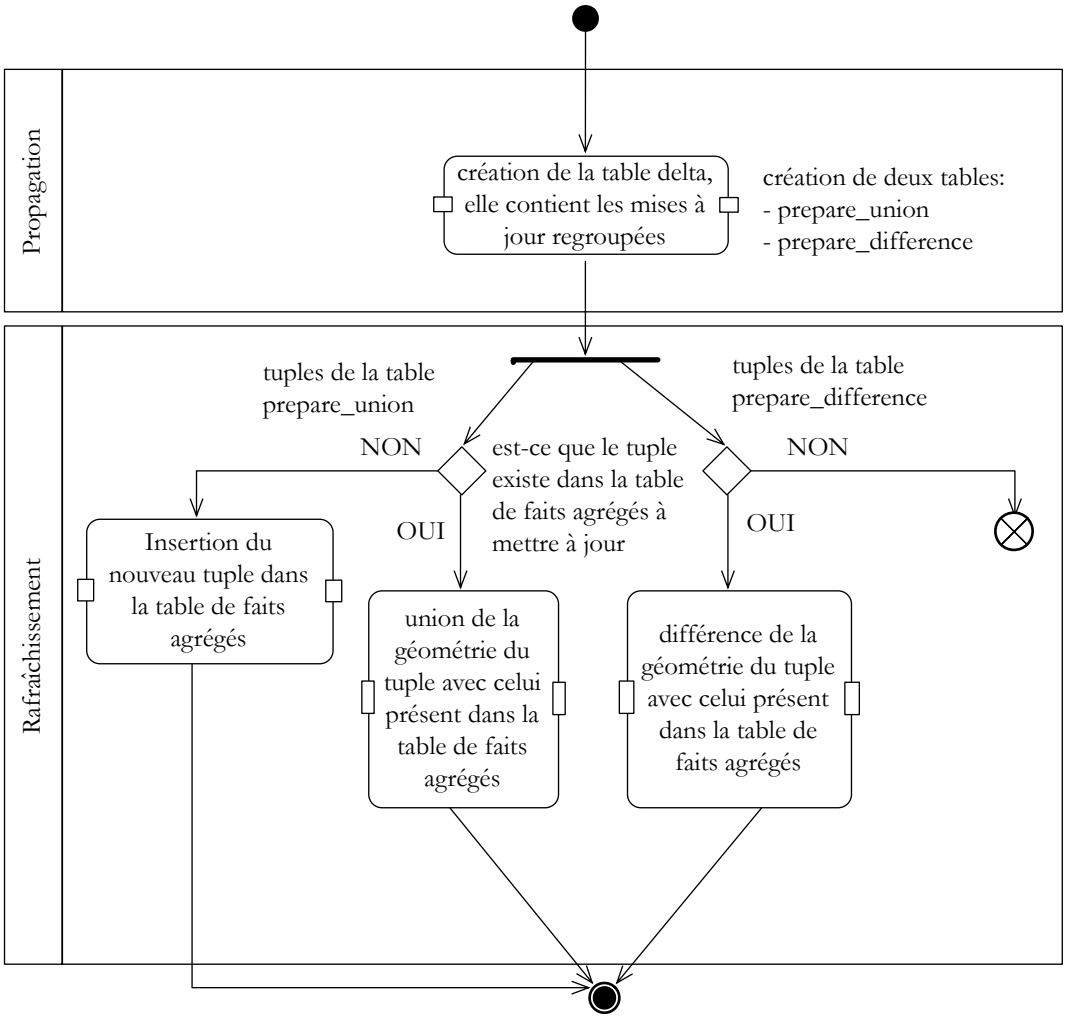


FIG. 4.10 – Diagramme UML d'activité de l'algorithme Spatiale Summary-delta.

La jointure est faite au niveau de la table de faits.

4.3 Mise à disposition du processus interopérable à distance

Les systèmes informatiques autrefois organisés de manière centralisée (client-serveur) évoluent vers des architectures distribuées (multitiers). Les logiciels installés sur un ordinateur font de plus en plus appel à des fonctions distantes, disponibles en

ligne. Certains logiciels sont entièrement en ligne, accessibles à l'aide d'un navigateur internet (p. ex. le traitement de texte offert par Google). Les architectures orientées service[5] révolutionnent la manière de construire des applications distribuées ; bâties sous forme de services (les services web), les applications échangent des données et interagissent entre elles plus efficacement qu'auparavant [Newcomer, 2002] : de manière interopérable, sans intervention humaine et en temps-réel. Les applications échangent des messages standards en XML, pour l'invocation et la réponse du service. Le format XML permet l'échange de données complexes et indépendantes de la plate-forme utilisée. De plus, les ressources ne sont plus situées sur un seul serveur, mais réparties sous forme de services sur le réseau. Cette division permet de gagner en rapidité.
Les processus de gestion de l'entrepôt de données sont encore souvent installés au coeur de l'entrepôt [Kimball et Caserta, 2004] ; les rendre interopérables et disponibles en ligne constituerait un bel effort. C'est dans cette optique d'architecture orientée services que nous souhaitons construire un service web de mise à jour de cube.

4.3.1 Théorie sur les services web

4.3.1.1 Définitions

Le World Wide Web Consortium (W3C) définit la notion de service web comme : "a software system designed to support interoperable Machine to Machine interaction over a network". Dans une architecture SOA, les programmes communiquent sur internet à l'aide de messages XML (eXtensible Markup Language) standardisés, ce qui les rend indépendants du langage de programmation, du système d'exploitation, des plateformes matérielles et logicielles. On parle d'interopérabilité des systèmes.
Il existe différents standards pour les services web, ils définissent le format des messages échangés, les mécanismes de découverte du service et de sa publication, etc. Le standard le plus connu est le standard *WS-** défini par le W3C ; en géomatique l'Open Geospatial Consortium (OGC) a défini plusieurs standards dédiés à l'échange, la diffusion de données géospatiales (WMS pour Web Map Service, WFS pour Web Feature Service)

[5] *SOA* pour Service Oriented Architecture

et à l'exécution de traitements (WPS pour Web Processing Service).
La mise à jour de cube spatial est un traitement, un processus ETL. Le service permettant son exécution à distance doit prendre en entrée le nom du processus à exécuter et appelle ainsi le lancement de l'exécution. En retour, il renvoie certaines informations sur le déroulement de la procédure de mise à jour. Aucune donnée spatiale n'est échangée entre les différents acteurs du service web, aussi il apparaît judicieux de s'orienter vers les standards définis par le W3C. De plus les plates-formes pour déployer des services W3C sont stables et ont fait leurs preuves. Bien que l'on en trouve de plus en plus pour les services de type WMS et WFS (Deegree[6], MapServer[7] et GeoServer[8]), les plates-formes pour les services WPS sont peu nombreuses et encore à l'état de conception (52 ° North [Foerster, 2006]). Nous optons donc pour les spécifications énoncées par le W3C que nous explicitons dans la suite de cette section.

4.3.1.2 Les services web W3C

Les services web W3C, aussi appelés services SWU[9], sont basés sur trois spécifications :

* SOAP (Simple Object Access Protocol) : format pour les messages XML de requête et de réponse du service.
* WSDL (Web Service Description Language) : format pour la description des services, de leurs opérations, des messages utilisés, des types de données, des protocoles et de leur localisation.
* les annuaires UDDI (Universal Description, Discovery and Integration) : pour référencer les services web.

On distingue trois acteurs majeurs dans une architecture orientée services W3C (Fig. 4.11) :

1. le fournisseur : il implémente le service et le fournit sur le réseau.

[6] http ://www.deegree.org
[7] http ://mapserver.gis.umn.edu
[8] http ://geoserver.org
[9] Pour SOAP, WSDL et UDDI

2. le demandeur de services : il fait appel au service par des requêtes XML
3. le registre de services : annuaire des services

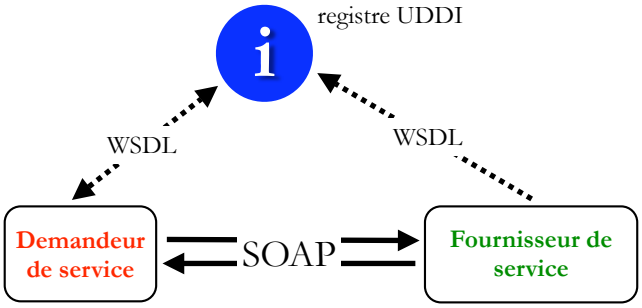


FIG. 4.11 – Architecture des services web W3C.

Le demandeur de service identifie le service à l'aide du registre qui lui envoie les informations nécessaires (fournies sous forme de fichier WSDL) pour l'utiliser. Ensuite l'appel et la réponse du service se font à l'aide de messages SOAP. SOAP est un protocole qui définit l'envoi d'un message XML par HTTP et la réception de la réponse. Un message XML SOAP contient : une enveloppe, une entête (optionnelle) et un corps de message (informations pour la requête ou pour la réponse). Des exemples de documents WSDL et SOAP sont donnés en Annexe D.

4.3.2 Description du contrat de service

Le contrat de service du service web est donné par le fichier WSDL. Il décrit le service et son fonctionnement de manière exhaustive et technique. Le contrat de notre service se situe donc en D. Il explicite dans l'ordre :

- les noms et formats des messages échangés
- les opérations fournies par le service
- les bindings : laisons entre les messages et les opérations
- l'adresse pour l'appel du service

Il convient d'expliquer les différentes opérations fournies par notre service avant de décrire les types de messages échangés. Les fonctionnalités d'un service sont implantées

sous forme d'opérations. Le service web de mise à jour de cube devra permettre de choisir et d'exécuter des procédures ETL de mise à jour. Le service web doit offrir des fonctionnalités de navigation dans l'arborescence de l'outil ETL pour identifier les procédures de mises à jour. Un outil ETL stocke les procédures de mises à jour (jobs) dans des répertoires, eux même contenus dans des référentiels, comme l'illustre la Figure 4.12. Afin d'exécuter une mise à jour, il faut connaître son nom et son emplacement précis dans l'arborescence de l'outil.

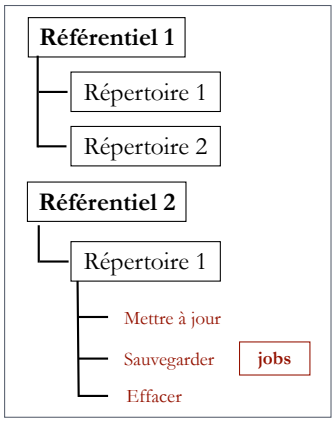


FIG. 4.12 – Architecture d'un outil ETL.

Le service conçu fournit donc trois opérations pour trouver les procédures : *listrep, listdir* et *listjobs*, et une opération pour leur exécution : *executeJob*. Les opérations *listrep, listdir* et *listjob* servent à naviguer dans l'arborescence de l'outil ETL pour trouver le job désiré. Une fois trouvé, l'usager appelle l'opération *executeJob* pour exécuter le job de mise à jour choisi. Le service web de mise à jour de cube se nomme *SOAP_Geokettle_list*, son fichier WSDL est fourni en Annexe D.1. Le diagramme de séquence des opérations du service est fourni en Figure 4.13.

De manière plus détaillée, les quatre opérations du service *SOAP_Geokettle_list* sont :

* ***listrep*** : retourne la liste des référentiels.
* ***listdir*** : prend en paramètres un nom de référentiel, un nom de connexion et un mot de passe. Retourne la liste des répertoires du référentiel.
* ***listjobs*** : prend en paramètres un nom de référentiel, un nom de connexion, un mot de passe et un nom de répertoire. Retourne la liste des jobs contenus dans le

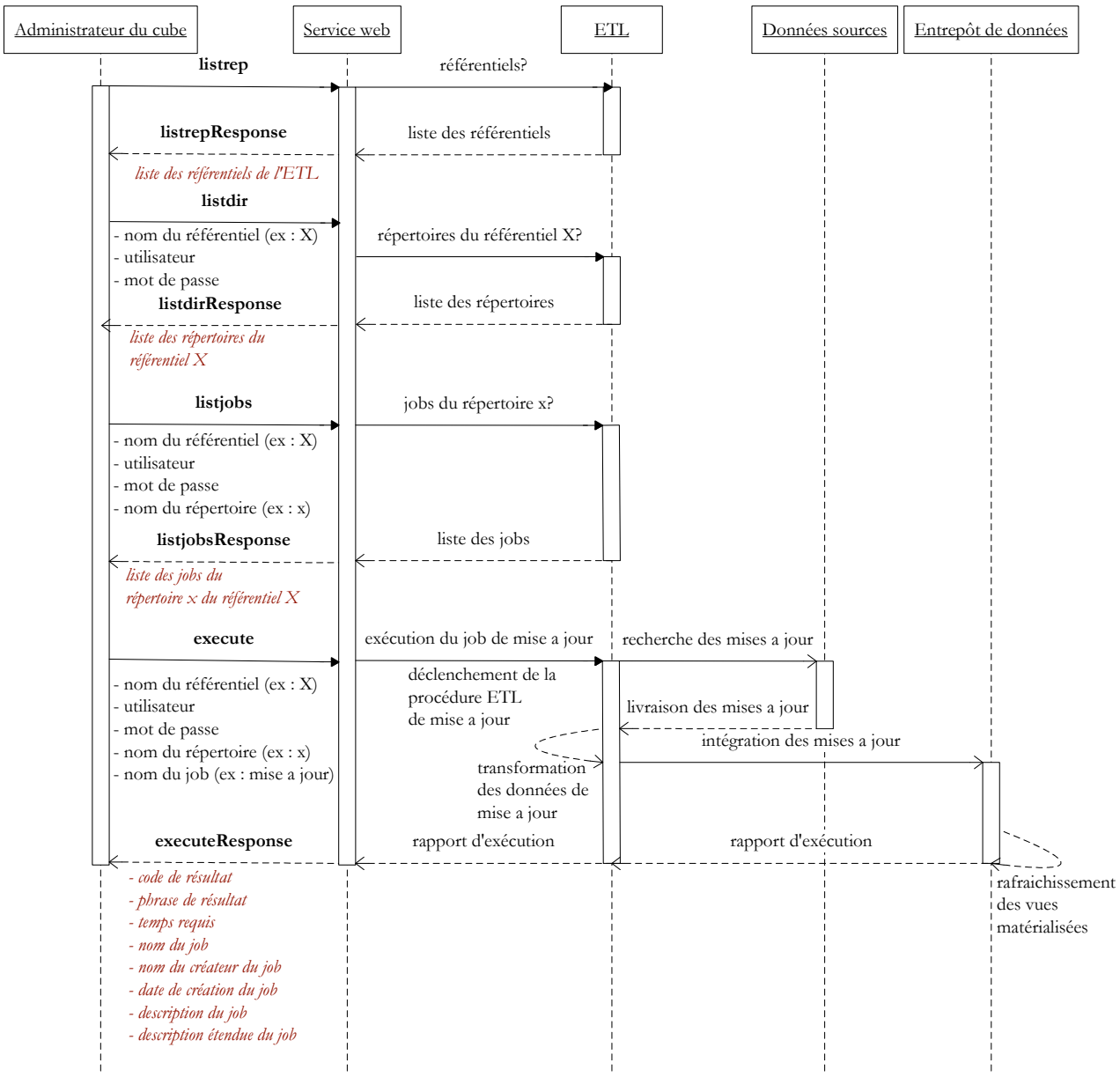


FIG. 4.13 – Diagramme de séquence des opérations du service web *SOAP_Geokettle_list*.

répertoire.

* ***executeJob*** : prend en paramètres un nom de référentiel, un nom de connexion, un mot de passe, un nom de répertoire et un nom de job.

  Exécute le job et retourne des informations concernant l'exécution (code de résultat, phrase de résultat, temps requis) et des métadonnées sur le job (nom, créateur, date de création, description et description étendue).

Les différents messages échangés seront donc :

** opération *listrep* :

* la balise *listrepRequest* ne contenant pas de message.
* la balise *listrepResponse* contenant un message de type chaîne de caractères et correspondant à la liste des référentiels trouvés.

** opération *listdir* :

* la balise *listdir Request* contenant les trois balises suivantes :
  - *repository* : le nom du référentiel dans lequel on cherche les dossiers.
  - *user* : le nom de l'utilisateur pour la connexion au référentiel (un référentiel est une base de données).
  - *password* : le mot de passe de l'utilisateur pour la connexion au référentiel.
- la balise *listdir Response* contenant un message de type chaîne de caractères et correspondant à la liste des dossiers trouvés.

** opération *listrep* :

* *listjobsRequest* contenant les quatre balises suivantes :
  - *repository* : le nom du référentiel dans lequel on cherche les dossiers.
  - *user* : le nom de l'utilisateur pour la connexion au référentiel (un référentiel est une base de données).
  - *password* : le mot de passe de l'utilisateur pour la connexion au référentiel.
  - *directory* : le nom du dossier à explorer.
* *listjobsResponse* contenant un message de type chaîne de caractères et correspondant à la liste des jobs trouvés dans le dossier.

** opération *listrep* :

* *listrepRequest* contenant les cinq balises suivantes :
  - *repository* : le nom du référentiel dans lequel on cherche les dossiers.
  - *user* : le nom de l'utilisateur pour la connexion au référentiel (un référentiel est une base de données).
  - *password* : le mot de passe de l'utilisateur pour la connexion au référentiel.
  - *directory* : le nom du dossier où se trouve le job.

- *job* : le nom du job à exécuter.

* *listrepResponse* contenant les huit balises suivantes :
  - *result_code* : le code de retour (0 : l'exécution s'est déroulée sans aucun problème ; 1 : l'exécution a rencontré des problèmes).
  - *result_phrase* : la phrase décrivant l'exécution (par exemple, pour une exécution sans problème : The job ran without problem).
  - *time* : la durée d'exécution du job.
  - *job_name* : le nom du job exécuté.
  - *creator* : le nom du créateur du job exécuté.
  - *creation_date* : la date de création du job exécuté.
  - *description* : la description brève du job exécuté.
  - *extended_description* :la description étendue du job exécuté.

Les balises de binding servent à relier les messages échangés à chaque opération. C'est en quelque sorte ce que nous venons de faire ci-dessus pour faciliter la description des messages.
Pour finir l'adresse d'appel de notre service sera :
http ://localhost :8080/axis/services/SOAP_Geokettle_listSOAP.

4.3.3 Limites du service

Actuellement, la gestion de l'entrepôt de données est une tâche réservée à l'équipe ETL, chaque membre de l'équipe va être chargé de la maintenance d'un système. Un membre de l'équipe, souvent l'administrateur de l'entrepôt de données, sera chargé de la mise à jour des cubes [Kimball et Caserta, 2004]. Avec une architecture SOA[10], il est maintenant possible de placer le service dans une architecture plus large. Par exemple en vue de l'automatisation du processus de mise à jour de cube et de son intégration avec d'autres processus, le service web de mise à jour de cube pourrait être appelé par une autre application ou un autre service. En résumé, le service web peut être appelé par des

[10]L'architecture de notre service sera détaillée en 5.4.

personnes, des applications ou d'autres services. Toutefois il ne doit pas être accessible aux usagers du cube, car la mise à jour nécessite la connaissance des procédures ETL et de l'architecture du cube et de l'entrepôt. On rejoint les propos de Kimball et Caserta [2004] sur l'analogie entre le restaurant et l'entrepôt de données. Les clients ne doivent en aucun cas accéder à ce qui se passe en arrière.

4.4 Conclusion du chapitre

L'intégration des données spatiales dans un cube pose des nouveaux problèmes pour la mise à jour. De nouveaux concepts entrent en ligne de compte : les mesures spatiales, les dimensions spatiales, les formats de données géométriques. Ces concepts sont récents et aucun standard ou méthode ne définit la manière dont doivent être intégrées les données spatiales dans le cube. Les cubes spatiaux existent principalement dans le domaine de la recherche scientifique, aucun produit commercial ne propose des cubes spatiaux contenant des mesures spatiales géométriques, certains proposent par contre d'effectuer des mesures spatiales numériques très simples (p. ex. Jmap SOLAP).
Tenant compte de ceci, nous avons défini des méthodes de mise à jour incrémentielles pour des types de cubes spatiaux simples[11]. L'établissement de contraintes d'intégrité géométrique nous a permis de décrire ces types de cubes pour lesquels nos méthodes seront valables. Nous avons ensuite classifié les types de mises à jour pouvant intervenir sur les données de dimension, et proposons une méthode pour chaque type de mise à jour. Seule la méthode de mise à jour proposée pour la modification de géométrie d'un membre de dimension fut testée lors de la recherche, son implantation et ses performances seront présentés en 5.3.2. Les méthodes pour la mise à jour des mesures spatiales numériques ont été explicitées et de nouvelles méthodes pour la mise à jour des mesures spatiales géométriques ont été définies. Ces dernières n'ont pas été implantées, faute de temps.
Souhaitant intégrer le processus de mise à jour de cube dans une architecture distribuée, nous avons conçu un service web de mise à jour. Nous avons expliqué la conception du

[11] De structure la plus basique : chaque dimension possède une hiérarchie stricte et balancée

service et ses fonctionnalités dans la dernière section.

Chapitre 5

Implantation et tests de performance des processus de mise à jour incrémentielle de cube et du service web

Les chapitres précédents ont détaillé les aspects théoriques du processus de mise à jour des cubes non spatiaux (cf. chapitre 3) et des cubes spatiaux (cf. chapitre 4). Le chapitre 5 présente le prototype du service web développé au cours de la recherche ainsi que l'implantation des méthodes définies dans les chapitres 3 et 4 dans un outil ETL. Pour réaliser ce prototype, nous avons utilisé plusieurs logiciels et avons conçu nos propres jeux de données pour les tests (section 5.1). Nous expliquons comment nous avons modifié le système pour faciliter les processus de mise à jour (section 5.2). Nous avons défini des procédures pour tous les types de mise à jour de cube non spatial établis dans notre typologie en 2.2.3.2, ainsi qu'une procédure pour la propagation incrémentielle des modifications de géométrie au sein de la dimension spatiale. La section 5.3 détaille l'implantation des procédures de mise à jour dans l'outil ETL et les méthodes de rafraîchissement des vues dans le SGBD. Les procédures de mise à jour de cube sont exposées à distance à l'aide du service web. L'architecture du prototype est décrite à

la section 5.4. Notre solution a pu être testée et validée, nos résultats sont présentés en section 5.5.

5.1 Choix technologiques d'implantation

Le prototype de service web de mise à jour de cube fait partie d'une architecture distribuée composée de plusieurs systèmes (cf. Fig. 5.1). Les sous-sections suivantes donneront des détails sur chacun de ces systèmes.

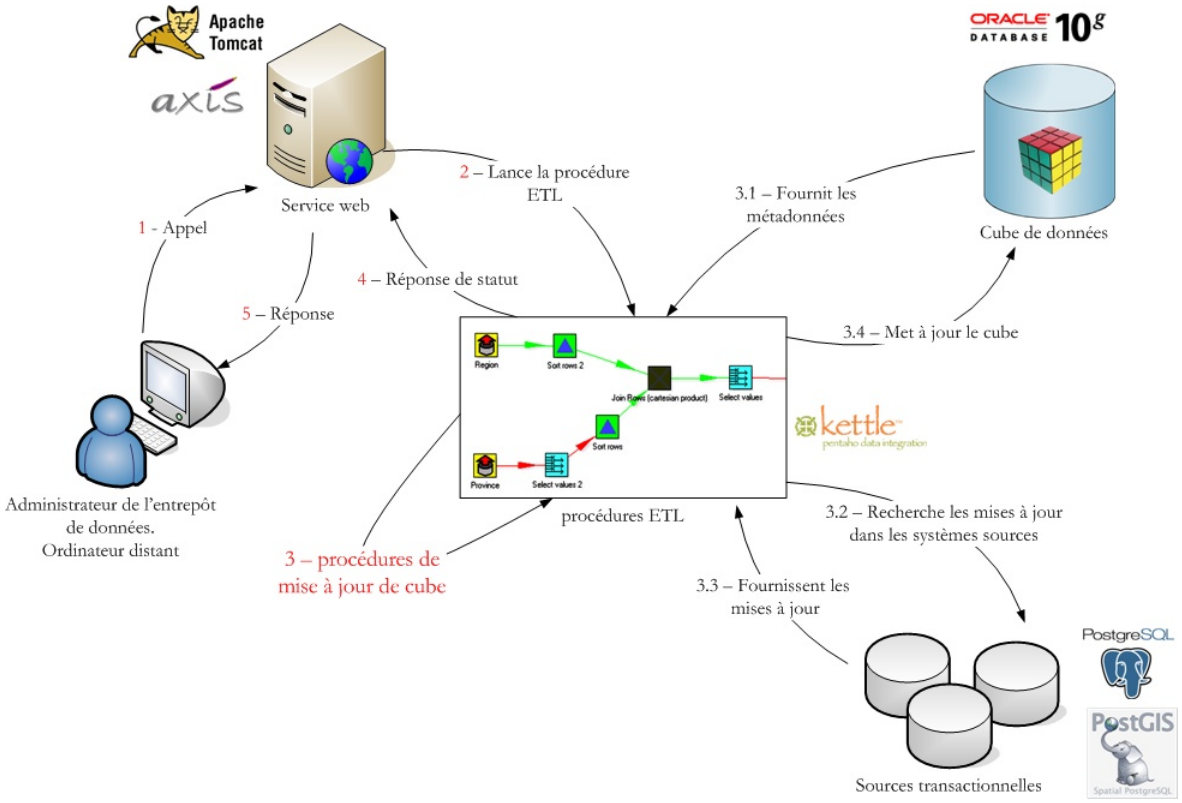


FIG. 5.1 – Architecture de la solution.

5.1.1 Constitution des jeux de données

Pour implanter nos procédures de mise à jour dans un outil ETL, nous avions besoin de plusieurs jeux de données. En ce qui concerne la mise à jour des cubes non spatiaux, il nous a dans un premier temps fallu un jeu de données simples, c'est-à-dire un petit

cube pour implanter nos procédures en testant leur fonctionnement. Ensuite, lorsque les procédures fonctionnaient, nous avons souhaité mesurer leurs performances en fonction du volume de mises à jour, mais également de la taille du cube à mettre à jour, et nous avions besoin pour cela de plusieurs cubes de données de tailles différentes. Nous avons procédé de même pour les procédures de mise à jour de la dimension spatiale : dans un premier temps, nous avions besoin de jeux de données spatiales simples (peu de points) pour implanter nos procédures. En effet, les tests sont beaucoup plus rapides avec des jeux de données simples. Ensuite, nous avons souhaité mesurer les performances de nos procédures de mise à jour en fonction du volume de mises à jour et du niveau de détail des données géométriques des membres de la dimension spatiale ; pour cela, nous avons besoin d'un jeu de données spatiales plus détaillées (centaines de points par polygone). Étant donné que notre projet de maîtrise s'intégrait dans un projet de recherche plus large : le projet GEOIDE visant à créer un Atlas Web pour analyser les impacts des changements climatiques sur la santé, il était donc normal d'examiner d'abord les données mises à disposition par ce dernier. Malheureusement, les jeux de données utilisés dans le projet GEOIDE sont des données statistiques par année, provenant de l'Institut National de Santé Publique du Québec et de Statistiques Canada, et également des simulations climatiques effectuées par l'organisme Ouranos pour plusieurs années. Ces données ne sont pas portées à varier fréquemment (tous les jours, semaines), mais plutôt tous les ans ou tous les 5 ans. Nous n'avons donc pas souhaité utiliser ces données pour nos tests, car la construction des cubes n'était pas assez avancée au moment où nous en avions besoin, et que le projet ne fournissait pas de mises à jour . Pour avoir des jeux de données plus adaptés aux cas de mise à jour, nous avons donc décidé de créer nos propres jeux de données. Nous avons donc imaginé trois cubes de données non spatiales pour tester nos procédures de mise à jour de cube non spatial :

1. un cube non spatial : ***sales_orders*** (cf. Annexe C, Fig. C.1 et C.2). Il contient au départ 1000 enregistrements dans la table de faits. Le cube représente les ventes fictives de certains produits pour certains clients. Ce cube est très petit (20 Ko) et nous fut utile pour concevoir nos procédures.
2. un cube non spatial représentant la population du Canada par divisions : ***po-***

pulation_divisions (cf. Annexe C, Fig. C.3 et C.5) utilisé dans un cours au département des Sciences Géomatiques de l'Université Laval. Les données proviennent de Statistiques Canada. Il contient environ 1 200 000 faits. Ce cube prend environ 1.4 Mo d'espace mémoire.

3. un cube non spatial représentant la population du Canada par provinces : ***population_provinces*** (cf. Annexe C, Fig. C.4 et C.6). Ce cube est obtenu à partir du précédent et contient environ 50 000 faits. Ce cube prend environ 600 Ko d'espace mémoire.

Les performances des procédures seront données par leurs durées d'exécution. La durée d'exécution d'une procédure de mise à jour est fonction du volume de mises à jour, mais également de la taille du cube à mettre à jour. Ainsi, ces trois cubes de tailles différentes vont nous permettre de comparer le temps d'exécution des procédures en fonction de la taille des cubes.

Nous avons également conçu deux cubes de données spatiales pour tester nos procédures de mise à jour de dimension spatiale :

1. un cube spatial représentant la population du Canada par provinces : ***population_provinces_spatial*** (cf. Annexe C, Fig. C.7 et C.8) avec une dimension spatiale représentant le Canada (cf. Annexe E, Fig. E.1), de géométrie très simple que nous avons dessinée dans ArcGIS : les polygones sont formés par peu de points. De même que le précédent, ce cube contient 50 000 faits, mais nous nous intéresserons principalement à sa dimension spatiale qui contient 12 provinces[1]. La taille du cube en espace mémoire est de 560 Ko.

2. un cube spatial représentant la population du Canada par provinces : ***population_provinces_spatial_detail***, même cube que le précédent, mais avec une dimension spatiale de géométrie très détaillée (cf. Annexe E, Fig. E.2). Les données spatiales proviennent des limites géopolitiques canadiennes fournies par la GeoBase[2]. Ce cube prend 4 Mo d'espace mémoire.

[1] Nous n'avons pas gardé la province de l'île du Prince Édouard pour nos tests.

[2] En libre téléchargement à l'adresse : http ://www.geobase.ca/geobase/fr/data/cgb/cgb1.html

Nous pensons que la durée d'exécution d'une procédure de mise à jour de dimension spatiale est fonction du volume de mises à jour, mais également du niveau de détail de la géométrie des membres de la dimension. Ainsi, ces deux cubes contenant des dimensions spatiales représentant les provinces du Canada de manière plus ou moins détaillée vont nous permettre étudier l'impact du détail de la géométrie des éléments sur les traitements spatiaux présents dans nos procédures de mise à jour de dimension spatiale.

5.1.2 SGBD source

Le SGBD source héberge les données sources transactionnelles, il doit donc pouvoir stocker les données spatiales. Plusieurs SGBD commerciaux fournissent la possibilité d'intégrer les données spatiales. Nous avons choisi d'utiliser le **SGBD OpenSource Postgresql**, muni de la cartouche spatiale **PostGIS**, pour stocker les données sources. Ce choix a été opéré, car nous souhaitions utiliser le plus possible des logiciels Open-Source pour la conception de notre prototype, ceci pour des raisons de coût et de philosophie de développement. De plus, ce SGBD est réputé pour ses bonnes performances par rapport aux solutions propriétaires.

5.1.3 SBGD hébergeant les cubes de données

Les cubes sont structurés en schéma étoile (le plus simple à mettre en oeuvre et le plus fréquemment utilisé) et nous avons décidé de matérialiser les agrégations sous forme de vues matérialisées. Aussi l'idéal est de choisir un SGBD qui gère nativement les vues matérialisées. Oracle 8i fut le premier SGBD à avoir implémenté les vues matérialisées. Les versions suivantes proposent désormais de nombreuses fonctionnalités pour gérer ces vues (cf. 3.3.3.3), Oracle 10g nous a donc paru être un bon choix.
Les cubes non spatiaux *sales_orders*, *population_divisions* et *population_provinces* nous serviront pour les tests sur la mise à jour des dimensions non spatiales, des tables de faits et des vues matérialisées. Ces trois cubes seront stockés dans une base de données

sous **Oracle 10g**.

Les cubes spatiaux *population_provinces_spatial* et *population_provinces_spatial_detail* nous serviront pour les tests sur la mise à jour des dimensions spatiales, ils ne contiennent pas de vues matérialisées. Ces deux cubes sont stockés dans le **SGBD Postgresql/PostGIS**.

5.1.4 Outil ETL

Il existe de plus en plus d'outils ETL OpenSource (Kettle, Talend Open Studio, Octopus, Clover.ETL, Ketl, etc.). Talend Open Studio de la société Talend gère depuis peu les données spatiales, mais est plus orienté vers les transformations de type transactionnelles sur les données spatiales, comme le logiciel ETL spatial FME. Kettle (maintenant Pentaho Data Integration) de la société Pentaho est un outil ETL conçu en Java. Il implémente les méthodes de *Slowly Changing Dimensions* et suit ainsi les recommandations de Ralph Kimball. De plus, une extension géospatiale à Kettle, nommée GeoKettle, a été développée au CRG par le groupe de recherche GeoSOA[3]. GeoKettle permet de manipuler les données spatiales au sein de l'outil ETL OpenSource. Pour extraire, transformer et charger les données dans le cube, nous utilisons l'outil **GeoKettle**.

Les principes de fonctionnement des différents outils ETL sont similaires : le flux de données emprunte un pipeline contenant des opérateurs de transformation. Les enregistrements sont traités indépendamment à la suite les uns des autres. Une procédure ETL est donc une chaîne d'opérations de transformation sur des données.

5.1.5 Service web

Comme nous l'avons vu précédemment (cf. 4.3.1.1), il existe deux acteurs majeurs de normalisation pour les services web : l'OGC pour les services web géospatiaux et le W3C pour les services web plus généraux. Le service web de mise à jour de cube

[3]http ://geosoa.scg.ulaval.ca

est un service fournissant un traitement, dans le monde géospatial, il s'apparenterait au service web de geo-processing : *Web Processing Service* (WPS) de l'OGC. Malheureusement, il n'existe actuellement qu'une seule plate-forme permettant de développer des services WPS [Foerster, 2006], encore en développement et peu documentée par rapport aux plate-formes de développement de services web SOAP (W3C). Ces dernières sont beaucoup plus nombreuses, mieux documentées et plus stables. La référence dans le domaine est Apache Axis[4]. Nous avons donc choisi d'utiliser la plate-forme **Apache Axis** pour concevoir et développer notre service web. Apache Axis est une implémentation des spécifications SOAP, c'est une application web déployée sur un serveur web. Nous optons pour la première version d'Axis (Apache Axis), il en existe une deuxième améliorée (Apache Axis2), mais la première est amplement suffisante et plus simple à mettre en oeuvre pour réaliser notre service. Le projet **Web Tools Platform** (WTP)[5] étend l'environnement de développement **Eclipse** en y intégrant une variété d'outils simplifiant le développement d'applications web. Notre service web sera développé et déployé sur Apache Axis à l'aide d'Eclipse WTP. Apache Axis sera hébergée sur le serveur web Tomcat[6], de la fondation Apache. Tomcat est le serveur web OpenSource le plus réputé et le plus utilisé, c'est le serveur web recommandé par le projet Apache Axis pour utiliser l'application Axis.

Les composants de notre prototype sont essentiellement des composants OpenSource[7]. Ces composants sont conformes à des normes et des standards ouverts, ce qui permet l'interopérabilité entre les systèmes (applications, services, clients, etc.). Le langage de programmation employé est le langage orienté objet Java, indépendant du système d'exploitation.

[4] http ://ws.apache.org/axis/
[5] http ://www.eclipse.org/webtools/
[6] http ://tomcat.apache.org/
[7] Sauf le SGBD Oracle.

5.2 Faciliter le processus de mise à jour

La mise à jour est un processus complexe qui dépend de beaucoup de paramètres (cf. 2.3.1). Dans le cadre de notre recherche, nous sommes administrateur de chaque partie de notre système, ce qui nous permet déjà de nous affranchir de nombreuses contraintes (disponibilité des sources, besoins des utilisateurs, etc.). Il est impossible de traiter tous les cas possibles de mise à jour (différents scénarios, différents types de cubes, différents types de données spatiales, etc.) ; nous émettons donc des restrictions (présentée dans la sous-section suivante) pour l'élaboration du prototype. Certaines parties du système peuvent également être modifiées et configurées spécialement pour faciliter le processus de mise à jour, c'est-à-dire l'accélérer et réduire la difficulté des algorithmes ou des processus mis en jeu.

5.2.1 Restrictions

Avant de présenter notre prototype et nos tests, il convient d'éclaircir les aspects que nous n'aborderons pas et dans quel cas nous nous plaçons pour mettre en oeuvre nos procédures de mise à jour.

Tout d'abord nous rappelons les contraintes d'intégrité sur lesquelles nous baserons notre prototype, énoncées en 4.1.2 :

* Les hiérarchies spatiales géométriques devront respecter la contrainte *full contains*.
* Les contraintes spatiales seront dures et nos hiérarchies à simple échelle.
* Toutes nos hiérarchies seront strictes et balancées.
* Les mesures spatiales numériques devront être distributives sur l'union.
* Nous ne considérons pas de superposition entre les membres d'un même niveau spatial.

Nous n'avons pas créé de procédure pour la mise à jour des dimensions volumineuses (*RCD*), nous choisissons les techniques de *Slowly Changing Dimensions*.

Au niveau des données spatiales, nous ne nous attaquons pas au problème de

généralisation cartographique, ni de multiples échelles. La généralisation cartographique et les représentations multiples au sein des bases de données à référence spatiale sont des sujets complexes, qui font toujours l'objet de recherches [Sabo et Bédard, 2007]. Il n'est pas de notre ressort de nous attaquer à ces problématiques. Aussi nous nous affranchissons des échelles de représentation pour nos dimensions spatiales : tous les niveaux de la dimension sont représentés avec le même niveau de détail peu importe l'échelle d'affichage. Les algorithmes *Spatial Summary-delta* resteront des propositions conceptuelles que nous n'avons pas eu le temps d'implanter et de tester. Les procédures de mise à jour ne traiteront donc pas de la mise à jour des mesures spatiales numériques et géométriques.

Nous étudions uniquement les géométries polygonales. Les processus pourraient être adaptés aux autres types de géométries assez aisément. La géométrie polygonale est pour l'instant le type de géométrie le plus implanté dans les cubes de données spatiales [Rivest *et al.*, 2005b].

Les processus ETL ne doivent pas gérer les index, il est conseillé de les détruire et de les reconstruire par la suite [Kimball et Caserta, 2004]. Nous avons décidé de ne pas aborder la reconstruction des index spatiaux. Nos procédures ETL ne la prennent donc pas en compte.

Un dernier point reste la livraison des données spatiales. Quel degré de confiance doit-on donner au fournisseur de données sur la fiabilité de ses données ; *est-ce que la mise à jour de la géométrie de la province Ontario doit être fournie lors d'une mise à jour sur la géométrie du Québec au niveau de la frontière avec l'Ontario ?* En d'autres termes, la source doit-elle contenir des données intègres et cohérentes ? Dans un certain sens, il serait logique que les sources de données spatiales soient intègres et cohérentes, par exemple *si la frontière entre le Québec et l'Ontario est modifiée, les géométries du Québec et de l'Ontario doivent être mises à jour au niveau des sources*, mais il est toujours préférable de vérifier l'intégrité des données sources. Nous étudierons une première solution dans laquelle nous n'accordons pas de confiance au fournisseur, et une deuxième solution où nous faisons amplement confiance au fournisseur de données afin de nous affranchir de vérifications qui pourraient être très coûteuses en temps de calcul.

5.2.2 Enrichissement des sources pour faciliter l'extraction des mises à jour

Réussir à capturer uniquement les données qui ont changé est un véritable challenge pour l'ETL (cf. 3.1). Nous avons recensé les techniques existantes pour l'extraction des mises à jour dans le Tableau 3.1.

Dans le prototype, l'extraction des mises à jour sera faite en **différé** et **par lots**. C'est l'ETL qui va chercher les mises à jour, donc il s'agit d'une méthode PULL [Kimball et Caserta, 2004]. L'ETL doit disposer d'indications pour trouver les mises à jour, aussi nous avons ajouté des ***timestamp*** dans les données sources pour indiquer la date de saisie de la mise à jour (***date de transaction***), et sa date de début de validité (***date de validité***). Ces deux dates nous permettent de distinguer les mises à jour tardives des mises à jour récentes.

Prenons l'exemple donné en Figure 5.2.

1 juillet 2007

customer_number	customer_name	customer_city	customer_state	transac_time	valid_time
1	Jean	Québec	QC	2007-07-01	2007-07-01
2	Pierre	Ottawa	ON	2007-07-01	2007-07-01
3	Paul	Montréal	QC	2007-07-01	2007-07-01

1 juillet 2008

customer_number	customer_name	customer_city	customer_state	transac_time	valid_time
1	Jean	Calgary	AB	2008-07-01	2008-07-01
2	Pierre	Toronto	ON	2008-07-01	2008-03-15

FIG. 5.2 – Exemple d'enrichissement des sources : la dimension *customer* du cube *sales_orders*.

Le 1er janvier 2007, des mises à jour sont saisies dans le système source transactionnel : Jean a déménagé le 1er juillet 2007 à Québec, Pierre à Ottawa et Paul à Montréal. Le 1er juillet 2008, de nouvelles mises à jour sont saisies : Jean est désormais à Calgary depuis le 1er juillet 2008 et Pierre habite Toronto depuis le 15 mars 2008, mais cette mise à jour a été saisie le 1er juillet. Si le cube a déjà été mis à jour entre le 15 mars

2008 et le 30 juin 2008, alors le dernier enregistrement sera une mise à jour tardive pour la prochaine mise à jour du cube du 1er juillet 2008.

5.2.3 Enrichissement du cube

Il est possible d'inclure des informations supplémentaires dans le cube de données afin de faciliter la détection (au niveau des sources) et la classification des mises à jour, la gestion de l'historique et la mise à jour incrémentielle des vues matérialisées. Ce sont des informations techniques qui vont servir à la mise à jour du cube et ne seront pas utiles aux usagers.

Détection des mises à jour

La détection des changements est gérée par l'ETL qui interroge simultanément les sources et le cube de données. Lors de la **n***ieme* mise à jour de cube, pour des mises à jour sur les données de la dimension X (resp. des faits), nous avons besoin de plusieurs informations afin de déterminer s'il s'agit de mises à jour récentes ou tardives (cf. Fig. 2.6 et Fig. 2.7), et de déclencher les procédures adaptées en conséquence. Ces informations sont :

- la date de changement dans la réalité : $\mathbf{T_{chgt\ n}}$. Elle est renseignée par le champ ***date de validité*** dans les tables sources.
- la date de saisie des mises à jour dans le système transactionnel : $\mathbf{T_{saisie\ n}}$. Elle est renseignée par le champ ***date de transaction*** dans les tables sources.
- la date de la dernière mise à jour du cube ayant pris en compte la ou les tables correspondant à la dimension : $\mathbf{T^{dimensionX}_{maj\ (n-1)}}$ (resp. aux faits, $\mathbf{T^{faits}_{maj\ (n-1)}}$). Cette dernière information doit être stockée dans le cube.

Pour conserver l'information $\mathbf{T^{dimensionX}_{maj\ (n-1)}}$ (resp. $\mathbf{T^{faits}_{maj\ (n-1)}}$), une **dimension d'audit** est créée. La dimension d'audit est intégrée au schéma étoile du cube, elle contient une clé primaire (SK), et la date de mise à jour associée à la table correspondante dans le schéma étoile du cube. Un exemple est donné en Figure 5.3.

audit_sk	last_update_date	table_updated
261	01/01/2007	customer_dim
262	01/01/2007	product_dim
263	13/01/2007	date_dim
264	13/01/2007	order_dim
265	13/01/2007	sales_order_fact

FIG. 5.3 – Extrait des enregistrements de la dimension d'audit du cube *sales_orders* après une mise à jour du cube.

Gestion de l'historique

L'historique des modifications peut être géré au sein de la table des faits avec la technique de *Rapidly Changing Dimensions* ou directement dans la dimension avec la technique de *Slowly Changing Dimensions* de type 2 (cf. 3.2.1). La technique *SCD2* partitionne l'historique par versionnement d'occurence, en ajoutant des colonnes de dates de validité dans la table de dimension :

- date de début de validité de l'occurrence
- date d'expiration de l'occurrence

Mise à jour incrémentielle des vues matérialisées

Il est beaucoup plus simple de mettre à jour incrémentiellement des vues qui sont *self-maintainable* (cf. 3.3.3.3). Les vues définies à l'aide de fonctions distributives et algébriques (p. ex. SUM, AVG, MIN et MAX) sont *self-maintainable* par rapport à l'insertion, mais pas par rapport à la suppression (MIN et MAX ne le sont pas).

Oracle pose certaines contraintes pour pouvoir utiliser la fonctionnalité de mise à jour incrémentielle des vues (*FAST_REFRESH*) :

- toutes les vues doivent contenir un champ COUNT(*) pour gérer les suppressions
- une vue contenant l'opérateur d'agrégation SUM(expression) doit contenir une colonne COUNT(expression)
- une vue contenant l'opérateur d'agrégation AVG(expression) doit contenir une colonne COUNT(expression) et une colonne SUM(expression)

5.2.4 Augmenter le débit de l'ETL

Les outils ETL sont paramétrables et certaines options permettent d'améliorer la rapidité des procédures. Afin d'augmenter le débit des données, il faut réduire le nombre d'éléments en entrée/sortie de la procédure, réduire la lecture/écriture dans les fichiers ou bases de données et éliminer le trafic réseau pour laisser le plus de bande passante à l'ETL. Nous nous sommes préoccupés des deux premiers points :

* Afin de réduire le nombre d'éléments en entrée/sortie de la procédure, nous effectuons l'extraction des mises à jour et le chargement dans le cube par lots[8] (pour prendre en compte le plus de mises à jour en effectuant le moins de déclenchements possible).
* Afin de réduire la lecture/écriture dans les fichiers ou bases de données, nous avons ajouté des index dans les tables source, et nous autorisons l'ETL à mettre en cache[9] le plus de rangées possible pour éviter les lectures multiples dans les sources.

De plus, l'ETL Kettle effectue tous ses traitements en parallèle, ce qui permet d'augmenter la rapidité d'exécution de la mise à jour.

5.3 Implantation des processus de mise à jour de cube dans l'outil ETL

Les processus de mise à jour sont implantés dans l'outil ETL OpenSource Kettle, fourni par la société Pentaho. Nous utilisons la version modifiée GeoKettle pour la gestion des données spatiales [Dubé *et al.*, 2007a].

GeoKettle dispose d'une interface graphique pour éditer les jobs et les transformations qui forment les procédures de mise à jour :

– une **transformation** contient plusieurs étapes (p. ex. extraction, jointure, insertion dans une table, etc.). Un exemple de transformation est fourni en Annexe F,

[8] Dans GeoKettle, il s'agit de donner une taille pour la validation de la transaction

[9] Dans GeoKettle, on peut paramétrer la taille du cache pour chaque étape

Figure F.1.

- un **job** assemble plusieurs transformations ou jobs. Un exemple de job est fourni en Annexe F, Figure F.4.

Nos procédures de mise à jour sont implantées sous forme de job : donc pour mettre à jour un cube, il suffit de lancer le job de mise à jour du cube.

Dans cette section nous expliquons comment nous avons implanté nos procédures de mise à jour de cubes non spatiaux et de cubes spatiaux dans GeoKettle. Toutes les dimensions sont mises à jour selon la technique *SCD2*.

5.3.1 Les types de procédures non spatiales

Une typologie complète pour la mise à jour de cube a été établie en 2.2.3.2, les différents types de mise à jour sont :

- les mises à jour récentes dans les dimensions
- les mises à jour récentes dans la table de faits
- les mises à jour tardives dans les dimensions
- les mises à jour tardives dans la table de faits

La mise à jour incrémentielle du cube se fait en plusieurs étapes (toutes de type incrémentiel) et respecte un ordre d'exécution précis énoncé en 2.2.4 et rappelé ici :

1. intégration des modifications tardives sur les dimensions
2. intégration des vieux faits
3. intégration des modifications récentes sur les dimensions
4. intégration des nouveaux faits
5. rafraîchissement des agrégations

Les étapes de la mise à jour de cube sont décrites par le diagramme UML d'activité illustré en Figure 5.4 et sont explicitées dans les sous-sections suivantes.

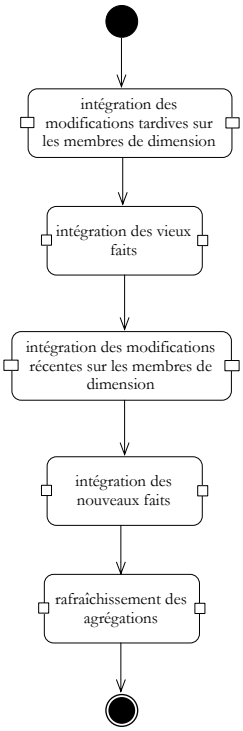


FIG. 5.4 – Diagramme d'activité représentant les étapes de la mise à jour de cube.

5.3.1.1 Mises à jour tardives dans la dimension

La saisie de mises à jour tardives est assez rare, mais il est nécessaire d'avoir une procédure prête pour ce type de mise à jour. Cette procédure est complexe. Nous l'avons implantée dans GeoKettle et testée pour la dimension *customer* du cube *sales_orders* (cf. Annexe F, Fig. F.4). Les étapes de la procédure sont décrites par le diagramme UML d'activité illustré en Figure 5.8.

Son principe de fonctionnement est le suivant :

- identifier la date de dernière mise à jour de la dimension *customer* (*last_update_date*) en lisant la table d'audit
- sélectionner les mises à jour tardives : celles dont $valid\_time < last\_update\_date < transac\_time$
- si le membre existe déjà, trouver l'occurrence antérieure la plus proche et mettre à jour sa date d'expiration. Ensuite trouver l'occurrence future et mettre à jour sa date de début de validité.

– affecter la clé étrangère de la nouvelle occurrence aux faits concernés (ayant lieu durant la période de validité de cette occurrence)
– recalculer les versions à l'aide d'une procédure stockée dans le SGBD cible (cf. Annexe G).

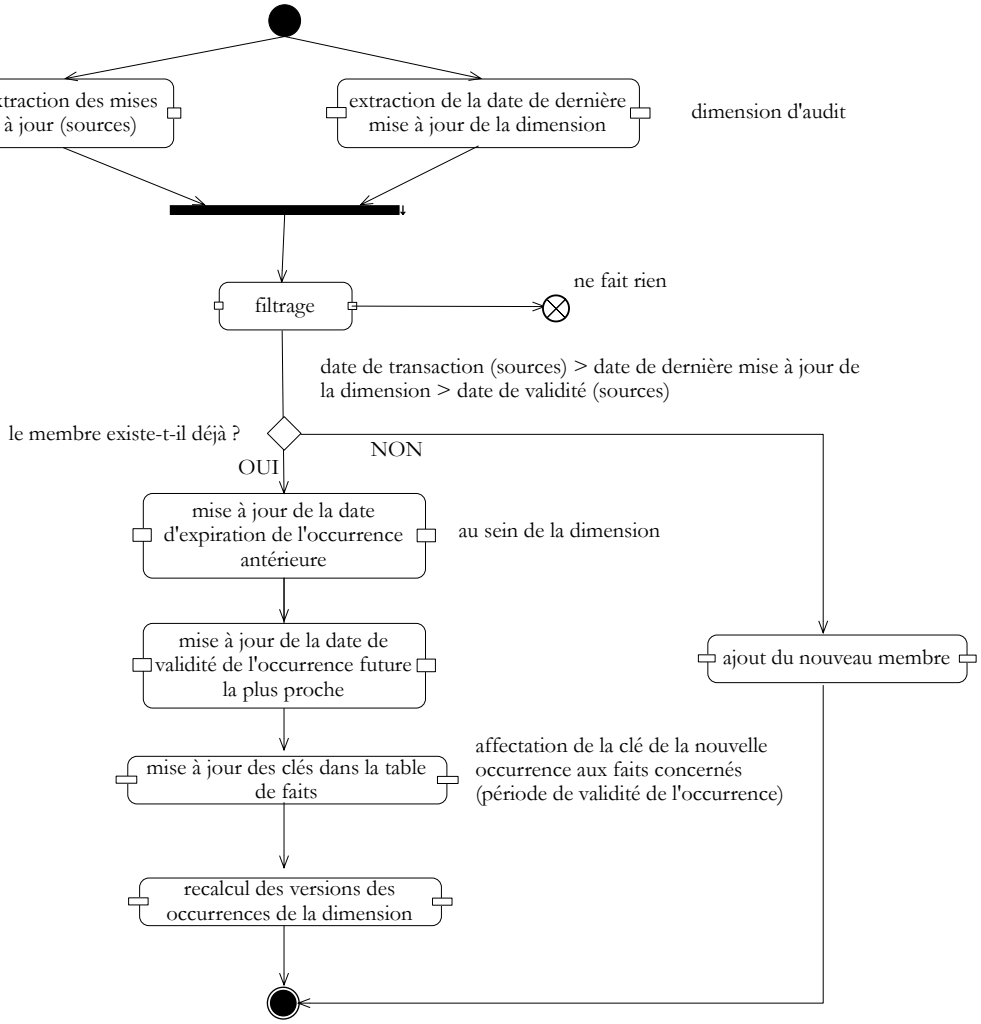


FIG. 5.5 – Diagramme d'activité de la procédure d'intégration des mises à jour tardives dans une dimension.

Les mises à jour tardives dans la dimension vont nécessiter le rafraîchissement incrémentiel des agrégations, mais celui-ci est effectué lorsque tous les types de mises à jour auront été saisis.

5.3.1.2 Mises à jour tardives dans la table de faits

La saisie des mises à jour tardives dans la table de faits se fait avec la même procédure que pour les mises à jour récentes dans la table de faits (voir suite). Prenons l'exemple du cube *sales_orders*, admettons que l'on saisisse des ordres pour le 15 janvier 2007 un mois en retard (nous sommes en février et le cube a déjà été mis à jour plusieurs fois), la procédure va chercher les occurrences correspondantes aux éléments de dimension dans les tables de dimension et valides au 15 janvier 2007. Pour cela il faut que le 15 janvier soit compris entre la date de début de validité et la date de fin de validité de l'occurrence.

5.3.1.3 Mises à jour récentes dans la dimension

Pour faciliter l'implantation de cette procédure, nous considérons que les sources ne subissent jamais de suppression, sauf en cas de mort d'un membre. Les mises à jour récentes peuvent être :

- une naissance de membre
- une modification sur un membre existant
- une mort de membre

Les étapes de la procédure sont décrites par le diagramme UML d'activité illustré en Figure 5.6. La procédure complète programmée dans GeoKettle se trouve en Annexe F, Figure F.1.

Dans le cas de modifications récentes, les procédures de mise à jour incrémentielles des dimensions ont été testées sur les dimensions *customer* et *product* du cube *sales_orders*. Le principe est le même pour les deux dimensions. La naissance de membre et la modification sur un membre existant sont traités par une même procédure : pour la dimension *customer*, en comparant la date de saisie dans la table source et la date de dernière mise à jour de la dimension *customer*, l'ETL extrait uniquement les nouvelles rangées de la table source *source_customer* (cf. Fig. 5.2 pour sa structure), c.-à-d. soit des nouvelles données (naissances de membres), soit des nouvelles occurrences (modifications sur un membre existant de la dimension). Ensuite à l'aide d'une étape spécifique

aux dimensions *SCD2*, la dimension est mise à jour :

- dans le cas d'une naissance de membre, une première version du membre est créée avec la date de début de validité du membre et une date d'expiration infinie.
- dans le cas d'une modification sur un membre existant possédant déjà n occurrences, une nouvelle occurrence du membre est créée avec un numéro de version égal à $n + 1$. La date d'expiration de l'occurrence numéro n est mise à jour, elle est égale à (date de début de validité de la version $n + 1$) - 1 (dans l'unité de temps du cube).

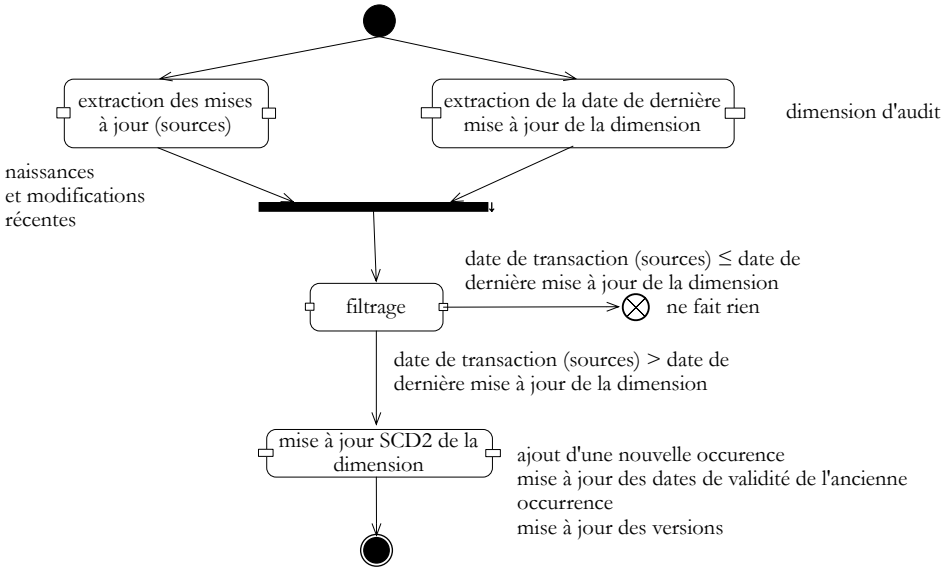


FIG. 5.6 – Diagramme d'activité de la procédure d'intégration des mises à jour récentes dans une dimension dans le cas de naissances ou de modifications de membres.

La mort d'un membre est traité par une procédure spécifique : en comparant les occurrences courantes de la dimension avec les occurrences dans la table source, si aucune occurrence n'existe dans la source, c'est que le membre est mort : on met à jour la date d'expiration de la dernière occurrence du membre.
Les étapes de la procédure sont décrites par le diagramme UML d'activité illustré en Figure 5.7. La procédure se trouve en Annexe F, Figure F.2.

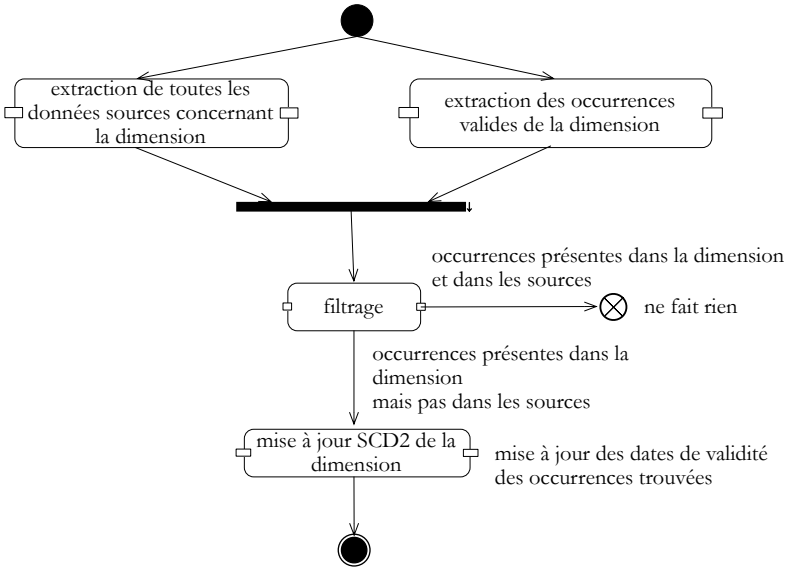


FIG. 5.7 – Diagramme d'activité de la procédure d'intégration des mises à jour récentes dans une dimension dans le cas de morts de membres.

5.3.1.4 Mises à jour récentes dans la table de faits

La saisie des mises à jour récentes dans la table de faits est la procédure de mise à jour la plus fréquente pour un cube de données. Souvent la mise à jour de cube ne consiste qu'en la saisie des mises à jour récentes dans la table de faits. Nous avons implanté cette procédure pour les cubes *sales_order*, *population_divisions* et *population_provinces*.
Les étapes de la procédure sont décrites par le diagramme UML d'activité illustré en Figure 5.8. La procédure se trouve en Annexe F, Figure F.3.

Le principe est le même pour chaque cube, nous l'expliquons ici pour le cube *sales_order* :
Les données sources sont contenues dans une seule table *source_sales_order* et contiennent les enregistrements des ordres d'achat avec la clé naturelle du client, du produit, la date et le montant. La procédure ETL doit rechercher les clés SK des enregistrements valides correspondants au client, au produit et à la date en effectuant des *lookup* sur les tables de dimensions. Elle génère ensuite une nouvelle clé d'audit et insère toutes les clés SK ainsi que le montant dans un nouvel enregistrement de la table de faits.

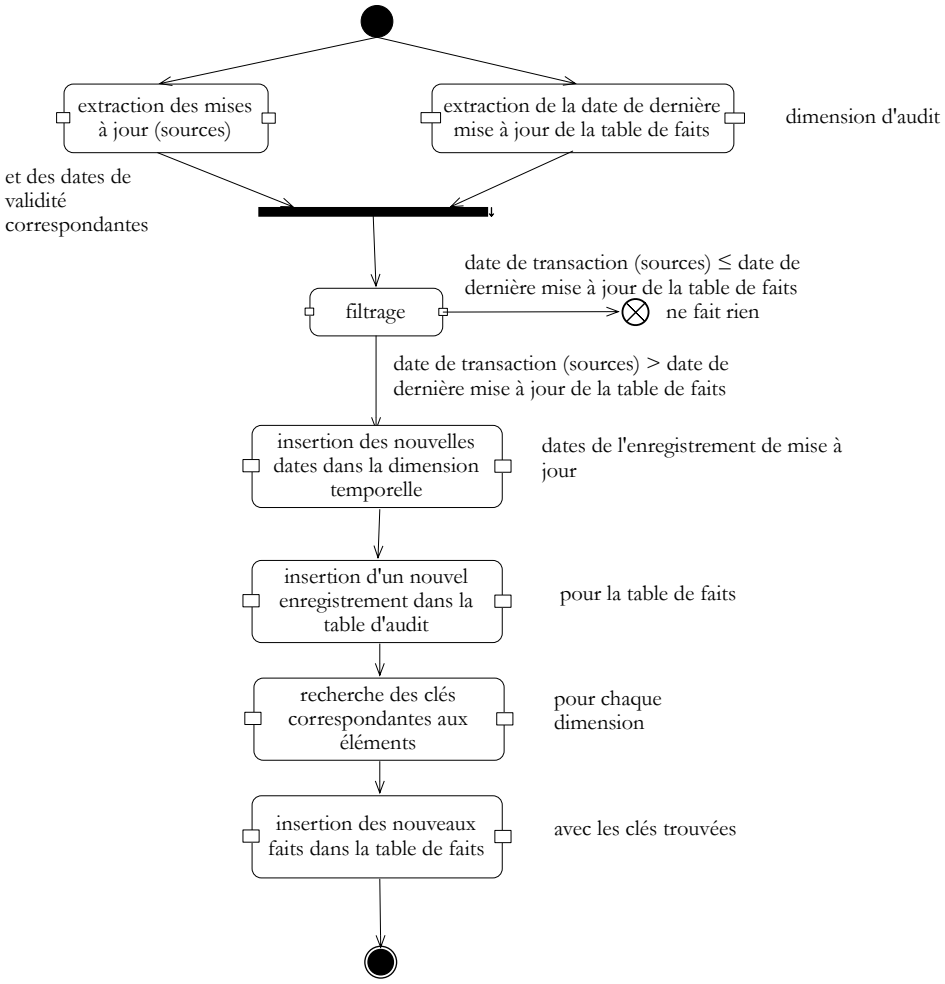


FIG. 5.8 – Diagramme d'activité de la procédure d'intégration des mises à jour récentes dans une table de faits.

5.3.2 La mise à jour des dimensions spatiales

Une typologie complète pour la mise à jour des dimensions spatiales a été établie en 4.2.1. Parmi les types de mises à jour possibles nous avons choisi de tester la **modification de la géométrie d'un membre détaillé** ; il s'agit pour nous du type le plus intéressant, car il faut détecter la modification de géométrie et gérer la mise à jour selon la technique *SCD2*. Intégrer les modifications de la géométrie des membres est plus complexe qu'ajouter un membre, ou supprimer un membre, et c'est une procédure de base qui peut être utilisée à plusieurs niveaux dans la dimension par les autres procédures (reclassification d'un membre, fusion et scission).

Si l'on considère la dimension spatiale du cube *population_provinces_spatial* et la mise à jour directe : modification de géométrie d'un membre détaillé, il existe quatre scénarios de propagation des mises à jour au sein de la dimension spatiale, comme le montre le Tableau 5.1 ; MD représente une modification directe, MI, une modification indirecte et 0, l'absence de modification. Le dernier cas (dernière ligne du tableau) est le plus fréquent, car, souvent, une modification de géométrie d'une ou plusieurs provinces va entraîner la modification des géométries des niveaux supérieurs.

province	**région**	**pays**	**explication**
0	0	0	aucune mise à jour de géométrie
MD	0	0	mise à jour de géométrie sur les provinces les géométries des régions et du pays ne sont pas touchées
MD	MI	0	mise à jour de géométrie sur les provinces les géométries des régions sont touchées, mais pas celles du pays
MD	MI	MI	mise à jour de géométrie sur une province les géométries des régions et du pays sont touchées cas le plus fréquent

TAB. 5.1 – Scénarios de propagation des mises à jour au sein de la dimension spatiale *province*

Plus précisément, nous choisissons donc de tester le dernier cas sur les deux cubes *population_provinces_spatial* et *population_provinces_spatial_detail* (cf. Annexe C, Fig. C.7 et C.8 pour le schéma conceptuel des cubes). Le niveau détaillé est le niveau province. Les géométries de tous les éléments de la dimension spatiale sont polygonales. La dimension spatiale *province* respecte les contraintes d'intégrité énoncées en 4.1.2, dont les contraintes *full contains* suivantes :

- $\bigcup_i province_i = \overline{regions}$
- $\bigcup_i region_i = \overline{pays}$
- $\bigcap_i province_i = \emptyset$
- $\bigcap_i region_i = \emptyset$

La table pays dispose d'un seul élément : le Canada.

Pour pouvoir implanter les procédures de mise à jour de dimension spatiale, nous avons d'abord adapté la structure des données sources en ajoutant les champs pour les dates de validité, ensuite nous avons imaginé une structure pour la dimension spatiale (cf. Fig. 5.9) : la dimension spatiale territoire est constituée d'une table centrale contenant toutes les données descriptives et de trois tables contenant les géométries des membres des trois niveaux hiérarchiques de la dimension (une table par niveau).

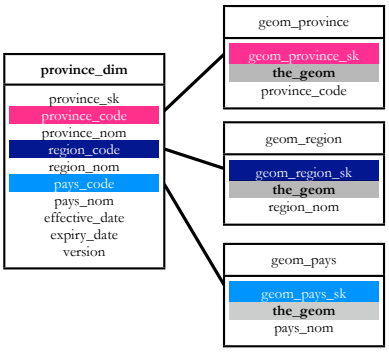
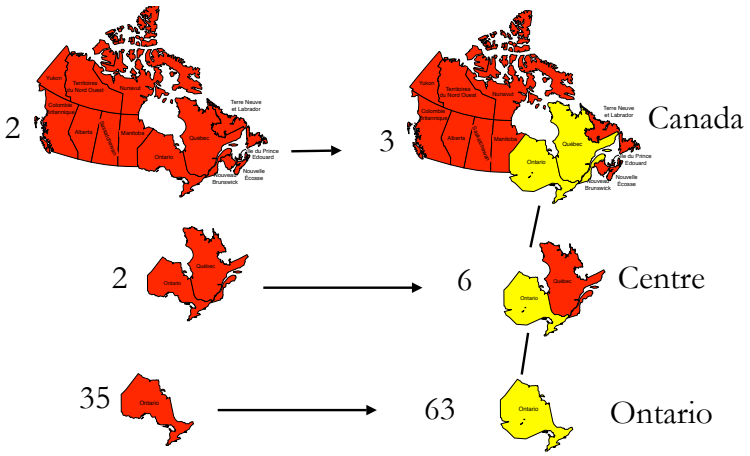


FIG. 5.9 – Dimension spatiale des cubes *population_provinces_spatial* et *population_provinces_spatial_detail*.

Nous avons procédé ainsi pour d'une part éliminer le plus possible la redondance, et d'autre part pour avoir un versionnement d'occurrence par niveau de hiérarchie. Les schémas de la Figure 5.10 montrent l'exemple d'une modification de la géométrie de la province de l'Ontario, cette modification implique un versionnement d'occurrence au sein de la dimension *province_dim* et un versionnement dans chaque table *geom_province*, *geom_region* et *geom_pays*. Nous avons créé nos propres sources de données pour disposer d'une géométrie simplifiée ; bénéficier d'une géométrie simplifiée (polygones d'une dizaine ou vingtaine de points) est plus pratique pour effectuer des tests et nous permet de comparer la rapidité des traitements avec d'autres géométries plus détaillées.

Une procédure de mise à jour de dimension spatiale est plus complexe qu'une procédure de mise à jour de dimension non spatiale : elle doit prendre en compte la propagation des modifications au sein de la dimension (cf. 3.4). Nous avons vu en 4.2.2 que cette propagation pouvait être traitée par analogie avec la propagation des mises à jour au

sk	province_nom	province_code	region_code	region_nom	pays_code	pays_nom	effective_date	expiry_date	version
10	Terre-Neuve-et-Labrador	NL	1	Atlantique	2	Canada	2007-10-02	2007-12-08	1
12	Nouvelle-Écosse	NS	1	Atlantique	2	Canada	2007-10-02	2007-12-08	1
13	Nouveau-Brunswick	NB	1	Atlantique	2	Canada	2007-10-02	2007-12-08	1
24	Québec	QC	2	Centre	2	Canada	2007-10-02	2007-12-08	1
35	Ontario	ON	2	Centre	2	Canada	2007-10-02	2007-12-08	1
46	Manitoba	MB	4	Prairies Territoire du Nord-Ouest et Nunavut	2	Canada	2007-10-02	2007-12-08	1
47	Saskatchewan	SK	4	Prairies Territoire du Nord-Ouest et Nunavut	2	Canada	2007-10-02	2007-12-08	1
48	Alberta	AB	4	Prairies Territoire du Nord-Ouest et Nunavut	2	Canada	2007-10-02	2007-12-08	1
59	Colombie-Britannique	BC	5	Pacifique Yukon	2	Canada	2007-10-02	2007-12-08	1
60	Territoire du Yukon	YT	5	Pacifique Yukon	2	Canada	2007-10-02	2007-12-08	1
61	Territoires du Nord-Ouest	NT	4	Prairies Territoire du Nord-Ouest et Nunavut	2	Canada	2007-10-02	2007-12-08	1
62	Nunavut	NU	4	Prairies Territoire du Nord-Ouest et Nunavut	2	Canada	2007-10-02	2007-12-08	1
63	Ontario	ON	6	Centre	3	Canada	2007-12-08	X	2



geom_province_sk	the_geom	province_code
1	MULTIPOLYGON (...)	35
2	MULTIPOLYGON (...)	63

geom_province

geom_region_sk	the_geom	region_code
1	MULTIPOLYGON (...)	4
2	MULTIPOLYGON (...)	6

geom_region

geom_pays_sk	the_geom	pays_code
1	MULTIPOLYGON (...)	2
2	MULTIPOLYGON (...)	3

geom_pays

FIG. 5.10 – Versionnement d'occurrence dans la dimension spatiale des cubes *population_provinces_spatial* et *population_provinces_spatial_detail*.

sein de la lattice des vues. Dans notre cas, la mise à jour de la dimension *province* comporte un volet non spatial et un volet spatial :

- mise à jour de la table non géométrique *province_dim* selon les techniques de mise à jour des dimensions non spatiales
- mise à jour des tables de géométries

Les différentes étapes de notre procédure sont, dans l'ordre :

- extraction des mises à jour
- détermination des types de modifications : géométriques ou non
- si la géométrie a été modifiée : appel de la procédure de mise à jour non spatiale pour la table *province_dim* et des procédures spatiales pour les tables de géométrie
- si la géométrie n'a pas été modifiée, on appelle uniquement la procédure de mise à jour non spatiale de la table *province_dim*
- mise à jour de la dimension d'audit
- archivage

Première solution : propagation pseudo-incrémentielle [10]

Les premières procédures implantées ne supposaient pas que les mises à jour sur les données sources étaient fournies intègres, aucune confiance n'était donc accordée sur ce point au fournisseur de données. Le processus de mise à jour devait donc se charger de vérifier l'intégrité du niveau détaillé *province* suite à l'intégration des mises à jour. Sur chaque niveau, si les contraintes n'étaient pas respectées par rapport aux frères (cf. Tab. 4.2), alors la procédure identifiait les frères avec lesquels il y avait un conflit et mettait à jour leur géométrie.

La procédure de mise à jour de la dimension spatiale *province* comprenait plusieurs transformations :

1. mise à jour de la table *province_dim* et de la table de géométrie de niveau *province*
2. propagation des mises à jour des géométries sur le niveau *province* (mise à jour des frères)

[10] reconstruction totale des géométries de niveau supérieur par analogie avec le recalcul des agrégats

3. propagation des mises à jour des géométries au niveau *région* par reconstruction totale du niveau (d'où le terme pseudo-incrémentiel)
4. propagation des mises à jour des géométries au niveau *pays* par reconstruction totale du niveau
5. mise à jour de la dimension d'audit
6. archivage

Geokettle utilise la plate-forme GeOxygene pour manipuler les objets géométriques. Les modifications de géométries, les vérifications du respect des contraintes d'intégrité se font à l'aide des opérateurs d'union, de différence symétrique, d'intersection et d'adjacence. Le module JavaScript de GeoKettle permet d'appeler toutes ces opérations fournies par GeOxygene[11].

Lors des tests, l'exécution de cette procédure s'est avérée extrêmement longue, en raison des multiples vérifications à effectuer au cours de la propagation des mises à jour de géométries sur le niveau *province*. La durée d'exécution de l'étape de vérification (2) constituait plus de 90% de la durée totale (1 min 50 s pour 2 min). Cette étape est nécessaire uniquement si les mises à jour ne sont pas fournies intègres et complètes. Si nous faisons confiance aux données sources, il ne semble pas utile de garder une étape si coûteuse. Nous avons donc envisagé une seconde solution ne prenant pas en compte les vérifications et améliorant la propagation des mises à jour des géométries aux niveaux supérieurs.

Seconde solution : propagation incrémentielle

La seconde solution reprend la première étape de la première solution : mise à jour de la table *province_dim* et de la table de géométrie de niveau *province*. Les étapes suivantes ont été modifiées : la mise à jour des géométries des couches supérieures (régions et pays) se fait désormais de manière incrémentielle (et non par reconstruction totale des géométries).

Il est possible d'établir un parallèle entre les opérateurs géométriques et les opérateurs standards SQL *insert*, *delete* et *update* que nous résumons dans le Tableau 5.2 :

[11] Une documentation plus détaillée de GeoKettle est disponible sur le site http ://www.geokettle.org

opérateurs SQL	**opérateurs géométriques**
insert	*union*
delete	*différence symétrique*
update=*delete* puis *insert*	*modification géométrique* = *différence symétrique* puis *union*

TAB. 5.2 – Équivalence entre les opérateurs SQL standards *insert*, *delete* et *update* et les opérateurs géométriques.

La mise à jour des niveaux supérieurs (région et pays) suit le principe établi par l'analogie suivante :

> Soient ***union_prov_del*** l'union des anciennes géométries des provinces mises à jour et constituant la région et ***union_prov_ins*** l'union des nouvelles géométries des provinces mises à jour et constituant la région. Suite à une modification de géométrie sur plusieurs provinces, on a :
>
> $$region = (region \triangle union\_prov\_del) \cup union\_prov\_ins \qquad (5.1)$$

Les éléments ***union_prov_del*** et ***union_prov_ins*** sont créés à la volée par requêtage sur la base de données sources, leur géométrie est formée à l'aide de l'opérateurs natif *geom_union* de PostGIS. La géométrie de l'élément ***region*** (Équation 5.1) est calculée à la volée avec le module JavaScript de Geokettle, à l'aide des opérateurs *union* et *différence symétrique* de GeOxygene.

Effectuer les opérations géométriques directement dans le SGBD est plus rapide que faire ces calculs avec les outils fournis par l'ETL. Dans la première solution, toutes les opérations géométriques étaient effectuées par la procédure ETL, alors que dans la seconde, certains calculs géométriques sont effectués par le SGBD source (mais toujours à la demande de l'ETL), ce qui permet un gain de temps considérable (pour la dimension spatiale avec les géométries simplifiées : 10 s contre plus de 2 min 30 s pour la première

solution[12]). Le désavantage de cette solution et qu'il devient difficile d'intégrer plusieurs sources de données provenant de diverses endroits pour former le cube.
La procédure de mise à jour incrémentielle de la dimension spatiale *province* est illustrée en Annexe F, Figure F.6. Elle comprend plusieurs transformations :

- mise à jour de la table *province_dim* et de la table de géométrie de niveau *province* (illustrée en Annexe F, Figure F.7)
- propagation des mises à jour des géométries au niveau *région* par modification géométrique du niveau (Équation 5.1, illustrée en Annexe F, Figure F.8)
- propagation des mises à jour des géométries au niveau *pays* par modification géométrique du niveau (même principe que pour le niveau région)
- mise à jour de la dimension d'audit
- archivage

5.3.3 Implantation des vues matérialisées

Les agrégations des cubes non spatiaux sont stockées sous forme de vues matérialisées dans le SGBD Oracle 10g. La mise à jour des vues est effectuée par le SGBD Oracle et non par la procédure ETL. Nous avons créé des vues matérialisées pour les trois cubes non spatiaux (mais pas pour les deux cubes spatiaux) : *sales_orders*, *population_divisions* et *population_provinces*. Nous avons vu en 3.3.3.3 qu'Oracle fournit plusieurs fonctionnalités pour manipuler les vues matérialisées. Aussi, parmi les options disponibles, nous choisissons de créer nos vues sur-le-champ et de ne pas les mettre à jour immédiatement après les mises à jour sur la table de faits du cube, mais à la demande (par appel de procédure), pour calculer le temps requis pour leur rafraîchissement. Afin de pouvoir mettre à jour (ou rafraîchir) la vue de manière incrémentielle, il est nécessaire de créer une table de *log* sur la table de faits. Cette table va enregistrer toutes les modifications effectuées sur la table de faits (insertions, suppressions et mises à jour), elle est vidée après chaque rafraîchissement

[12]rappelons que la seconde solution n'effectue aucune vérification géométrique, contrairement à la première

de la vue.

Un exemple de requête de création de vue est donné en Figure 5.11, toutes nos vues seront construites de cette manière. Pour chaque cube, nous avons créé trois vues matérialisées organisées sous forme de lattice. Dans Oracle, on parle de *Nested Materialized Views*. Dans un même cube, chaque vue est dépendante d'une autre vue de plus grande taille, il faut donc créer une table de *log* sur la vue qui alimente la vue dépendante. Les vues dépendantes seront rafraîchies de manière immédiate (*on commit*). Toutes les vues matérialisées créées sont illustrées en Annexe C.

```
CREATE MATERIALIZED VIEW LOG ON "CHARLOTTE"."A_SALES_ORDER_FACT"
  WITH ROWID, SEQUENCE ( "AUDIT_SK", "CUSTOMER_SK", "ORDER_AMOUNT", "
      ORDER_DATE_SK", "ORDER_SK", "PRODUCT_SK" ) INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW "CHARLOTTE"."AGG_SALES_ORDER_CS_PC_DY_MV"
  BUILD IMMEDIATE
  REFRESH FAST ON DEMAND
  AS SELECT customer.CUSTOMER_STATE, product.PRODUCT_CATEGORY, a_date_dim."YEAR"
      , SUM(s.order_amount)  amount, COUNT(s.ORDER_AMOUNT) AS cnt, count(*) as
      count_all
FROM A_SALES_ORDER_FACT s, A_CUSTOMER_DIM customer, A_PRODUCT_DIM product,
    A_DATE_DIM
WHERE s.CUSTOMER_SK = customer.CUSTOMER_SK
AND s.ORDER_DATE_SK = a_date_dim.DATE_SK
AND s.PRODUCT_SK = product.PRODUCT_SK
GROUP BY customer.CUSTOMER_STATE, product.PRODUCT_CATEGORY, a_date_dim."YEAR";
```

FIG. 5.11 – Requête de création d'une vue matérialisée sous Oracle 10g.

La mise à jour incrémentielle des vues se fait par l'exécution d'une requête du type :

```
execute DBMS_SNAPSHOT.REFRESH('agg_sales_order_cs_pc_dy_mv','f');
```

L'option *f* représente le type de rafraîchissement : ici *fast*, pour rafraîchissement incrémentiel. Il existe d'autres options, comme *c* pour un recalcul complet de la vue.

5.3.4 Procédures globales de mise à jour de cube

Chaque cube possède une unique procédure de mise à jour implantée sous forme de *job* dans Geokettle. Le job de mise à jour du cube peut contenir des jobs et transformations qui constituent une étape de la mise à jour. Nous n'avons pas implanté de procédure globale de mise à jour pour les cubes spatiaux, car nous voulions que les procédures globales comprennent le rafraîchissement des vues matérialisées et nos cubes spatiaux, stockés dans PostGIS ne contiennent pas de vues matérialisées. Les procédures de mise à jour des cubes spatiaux sont uniquement des procédures de mise à jour de dimension spatiale[13]. Nous décrivons ici dans l'ordre, le contenu du job de mise à jour pour chacun de nos trois cubes non spatiaux :

- ***sales_orders*** (cf. Annexe F, Fig. F.5), job de mise à jour de cube :
  1. Job de mise à jour de la dimension *customer* : mises à jour récentes et tardives
  2. Transformation pour les mises à jour récentes sur la dimension *product*
  3. Transformation pour la mise à jour de la dimension *audit*
  4. Job de mise à jour de la dimension *date* et de la table de faits, mises à jour récentes et tardives
  5. Transformation pour l'archivage
  6. Procédure Oracle pour le rafraîchissement des vues matérialisées
- ***population_divisions*** et ***population_provinces***, job de mise à jour de cube :
  1. Transformation pour la mise à jour de la table de faits, mises à jour récentes et tardives
  2. Transformation pour l'archivage
  3. Procédure Oracle pour le rafraîchissement des vues matérialisées

Nous avons détaillé le fonctionnement des différents types de procédures de mise à jour implantées dans l'outil ETL et le rafraîchissement des agrégations dans le SGBD

[13] Puisque nos cubes spatiaux possèdent la même table de fait et les mêmes dimensions non spatiales que le cube *population_provinces*, la procédure globale de mise à jour des faits serait exactement la même, il ne nous a donc pas paru nécessaire de l'implanter

Oracle. GeoKettle stocke les procédures de mise à jour dans un référentiel sur notre SGBD PostgreSQL. Notre service web va lire les procédures dans le référentiel pour les exposer sur internet et permettre leur exécution à distance. La prochaine section explicite la mise à disposition des procédures de mise à jour sous forme de service web.

5.4 La mise à disposition des processus de mise à jour sous forme de service web

Tel que nous l'avons discuté, nous avons choisi de mettre à disposition les processus de mise à jour de cube sous forme de service web. Les services web présentent en effet de nombreux avantages pour le déploiement d'architectures distribuées interopérables. Notre service web offre plusieurs fonctionnalités : il permet de connaître les référentiels, la liste des répertoires, la liste des jobs et d'exécuter un job. Il est destiné aux administrateurs du cube.

Développement du service

Le développement du service s'est fait à l'aide de l'environnement Eclipse, muni de la plate-forme des outils web[14]. Cette plate-forme facilite le développement de services web, notamment par l'écriture du fichier WSDL sous forme graphique. Le projet Eclipse de notre service doit également contenir les librairies fournies par Axis.

Un service web est constitué d'une classe nécessaire à son fonctionnement interne (l'interface *SOAP_Geokettle_listSOAPortType*, appelée par les clients), et d'une classe d'implémentation *SOAP_Geokettle_listSOAPImpl*. C'est la classe principale, *SOAP_Geokettle_listSOAPImpl*, qui redirige les appels vers les opérations, également implémentées sous forme de classe. La classe *SOAP_Geokettle_listSOAPortType* est générée automatiquement par Eclipse et Axis à l'aide du fichier WSDL. Il reste à écrire les classes d'implémentation des opérations, parties fondamentales nécessaires à l'exécution de chaque opération.

[14]Web Tools Platform (http ://www.eclipse.org/webtools/)

Dans notre service, chaque opération du service est implémentée par une classe : *listrep* par la classe *RunKitchenSOAP_listrep*, *listdir* par la classe *RunKitchenSOAP_listdir*, *listjobs* par la classe *RunKitchenSOAP_listjobs* et *executeJob* par la classe *RunKitchenSOAP*. Toutes ces classes se basent sur la classe java Kitchen. Kitchen est un programme Java fourni par librairie Kettle.jar qui permet d'exécuter les jobs conçus avec Spoon (programme permettant de concevoir les procédures ETL de manière graphique) sous forme de fichier XML ou stockés dans un référentiel. Pour développer notre service, nous nous sommes inspirés de parties de codes sources de la classe Java Kitchen.

Un service web est implémenté sous forme de classes contenues dans l'application Axis sur le serveur Tomcat. Une fois le service développé sous Eclipse, il faut donc déplacer les classes compilées dans l'application web Axis sur le serveur Tomcat. Le service web est ainsi déployé sur le serveur Tomcat, au sein de l'application Axis, et est disponible sur internet.

Le diagramme de classes du service web est disponible en Figure 5.12.

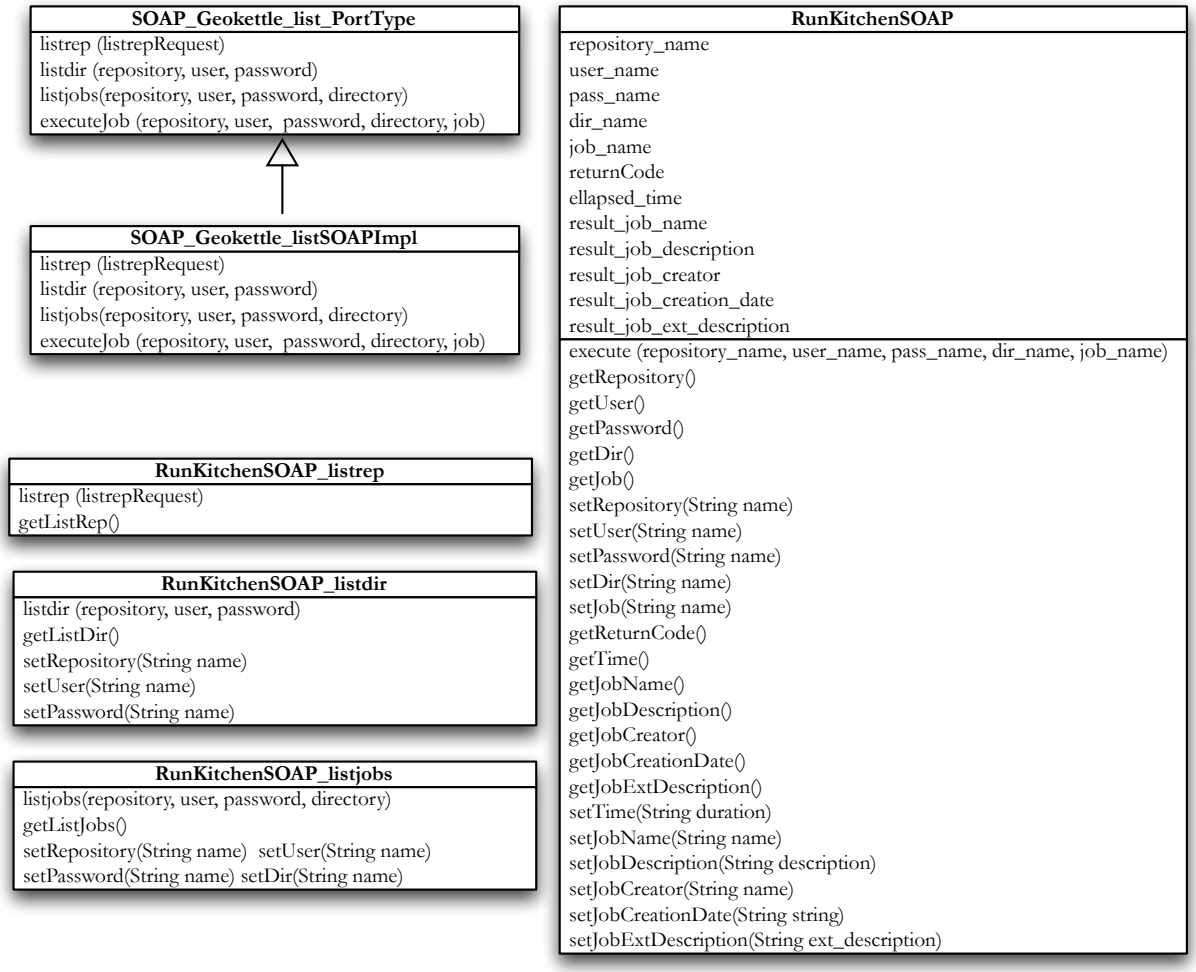


FIG. 5.12 – Diagramme de classe du service web *SOAP_Geokettle_list*

Fonctionnement du service

Le fonctionnement du service web pour l'exécution d'une procédure ETL est décrit en Figure 5.13 : un client envoie une requête SOAP au serveur demandant l'exécution d'une procédure. Pour cela il appelle la méthode *executeJob* avec les paramètres nécessaires. Le service web reçoit la requête, il la déchiffre et crée une instance de la classe d'implémentation du service : *SOAP_Geokettle_listImpl*. Il exécute la fonction *executeJob* de cette classe avec les différents paramètres passés en arguments. La réponse est ensuite formattée sous forme de message SOAP et renvoyée au client. Il s'agit ici d'une explication simplifiée de ce qui se passe en réalité à l'intérieur du serveur web.

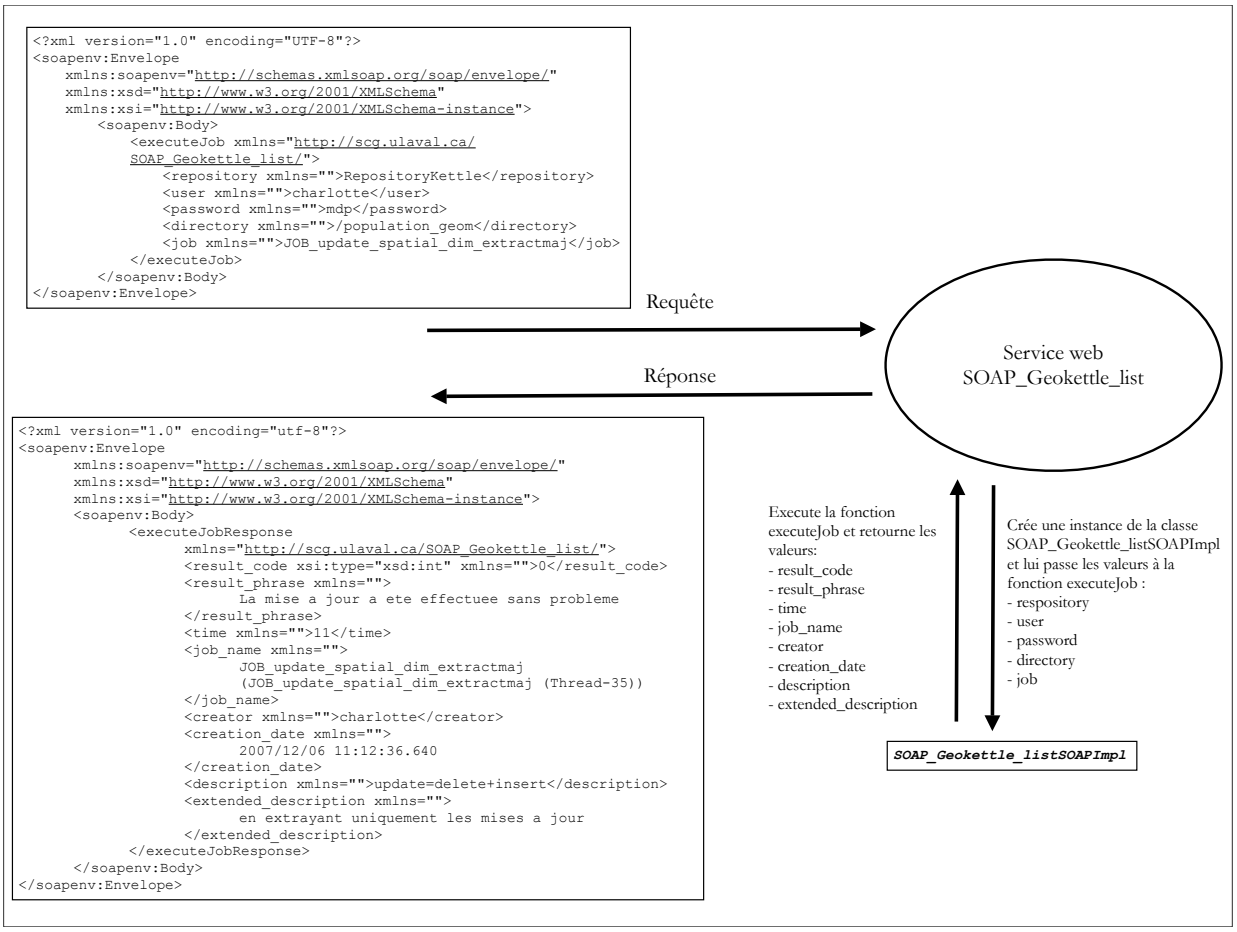


FIG. 5.13 – Fonctionnement du service web pour la fonction executeJob.

5.5 Tests de performance

Notre prototype de service web de mise à jour de cube contient deux parties majeures : les procédures ETL et le service web. Le service web a été validé sans tests de performance indépendamment des procédures de mises à jour, car l'élaboration de procédures de mise à jour de cube et la conception et le développement du service web qui va nous permettre d'appeler ces procédures à distance sont deux objectifs indépendants de notre maîtrise. Les tests de performance ne concernent pas la partie service web, mais les procédures ETL. La validation de nos méthodes (décrites en 5.3) dépend de la performance des procédures implantées dans l'outil ETL. Une fois les procédures implantées, il nous reste à vérifier leur performances selon différents critères :

– la durée d'exécution : les procédures de mise à jour incrémentielles devraient être

plus rapides que les procédures de mise à jour par reconstruction totale

- le nombre d'enregistrements traités : ce nombre doit correspondre au nombre attendu en fonction du volume de mises à jour
- le nombre d'enregistrements lus : ce nombre doit correspondre au nombre attendu en fonction du volume de mises à jour
- le nombre d'enregistrements écrits : ce nombre doit correspondre au nombre attendu en fonction du volume de mises à jour
- le nombre d'enregistrements traités ÷ durée (sec) × (nombre de bits / enregistrement)

Nous mesurons la performance de nos procédures de mise à jour selon le critère de durée d'exécution en fonction du nombre de mises à jour. GeoKettle indique également les autres statistiques énoncées ci-dessus lors de l'exécution d'un job ou d'une transformation.

Nous avons effectué des tests sur les cinq cubes présentés au cours de ce mémoire (cf. Annexe C) :

- **la mise à jour de cube** a été testée sur les trois cubes non spatiaux de différentes tailles : *sales_orders*, *population_divisions* et *population_provinces*
- **la mise à jour de dimension spatiale** a été testée sur les deux cubes spatiaux *population_provinces_spatial* et *population_provinces_spatial_detail* afin de voir les impacts du détail des géométries sur la performance des procédures de mise à jour.

Les tests ont été réalisés dans un environnement Mac Os X, sur un ordinateur muni d'une mémoire vive de 2 Go et d'un processeur Intel Pentium Core 2 Duo de fréquence 2.2 GHz.

5.5.1 Tests et validation du service web

L'important est de vérifier le bon fonctionnement du service web, c'est-à-dire la bonne exécution de l'opération demandée et la conformité des messages échangés par rapport aux standards. Pour cela, nous avons utilisé le cadre de développement Axis

pour implémenter une classe qui fait office de client et appelle le service web. L'exécution des opérations : *listrep*, *listdir*, *listjobs* et *executeJob* du service appelées par la classe cliente s'est bien déroulée : nous avons obtenu les réponses attendues. À l'aide d'un package fourni par Apache Axis (Axis TCP Monitor), nous avons observé les messages SOAP de requête générés par notre client et les messages de réponse renvoyés par notre service pour chaque opération. Ces messages sont conformes au standard SOAP et sont disponibles en Annexe D. Ces tests nous ont ainsi permis de valider le bon fonctionnement de notre service web : structure des messages échangés conformes aux attentes, bonne exécution des fonctions appelées, contenu des messages de retour conformes aux réponses attendues, etc.

5.5.2 Tests des procédures de mise à jour de cube non spatial

Pour chacun des trois cubes, nous avons mesuré le temps d'exécution des procédures de mise à jour de cube en fonction du nombre de mises à jour fournies. Une comparaison des performances est établie pour trois scénarios de mise à jour :

1. incrémentielle : mise à jour incrémentielle de la table de faits et des agrégations
2. pseudo-incrémentielle : mise à jour incrémentielle de la table de faits et reconstruction des agrégations
3. reconstruction totale du cube

Nous ne disposions pas de données réelles mises à jour assez fréquemment pour nos tests. Les données de mise à jour sont donc fictives. Les données de mise à jour servant à peupler les tables de faits ont été simulées à l'aide d'un programme écrit en langage Java. Ce programme attribue de façon aléatoire une valeur de clé servant à identifier chaque membre de dimension et des valeurs aléatoires pour les différentes mesures. Les données de mise à jour pour les dimensions ont été écrites à la main dans des fichiers SQL.

Il est à noter que plusieurs paramètres peuvent influencer le temps d'exécution des procédures de mise à jour :

– la procédure utilisée est différente pour chaque cube.

- le trafic réseau : la base de données Oracle n'était pas située en local
- l'occupation du processeur faisant tourner l'ETL (en local pour nous),
- l'occupation du processeur de la base de données sources (en local également)
- l'occupation du processeur de la base de données cible Oracle (notamment pour la mise à jour des vues).

Nous avons cherché à minimiser l'impact de ces paramètres en travaillant le plus possible en local, ainsi nous connaissons l'occupation du processeur et éliminons les fluctuations dues au trafic réseau. Toutefois, les résultats que nous avons obtenus sont propres à nos cubes et nos procédures, et ne doivent en aucun cas être interprétés comme des résultats génériques.

Cube *sales_orders* (Annexe C, Figures C.1 et C.2)

Le cube initial contient 1 000 enregistrements dans la table de faits, il comporte trois vues matérialisées. Nous avons testé la rapidité d'exécution de la mise à jour de cube pour des lots de mises à jour de volumes différents variant de 1 000 à 10 000 enregistrements, par incréments de 1 000. Les résultats sont affichés en Figure 5.14.

La méthode incrémentielle n'est pas extrêmement plus rapide que la reconstruction totale du cube. La plus grande partie du temps d'exécution est due à l'enchaînement des étapes ETL plus qu'aux traitements des données. La reconstruction des vues prend autant de temps que leur mise à jour incrémentielle (la courbe verte est cachée par la courbe bleue).

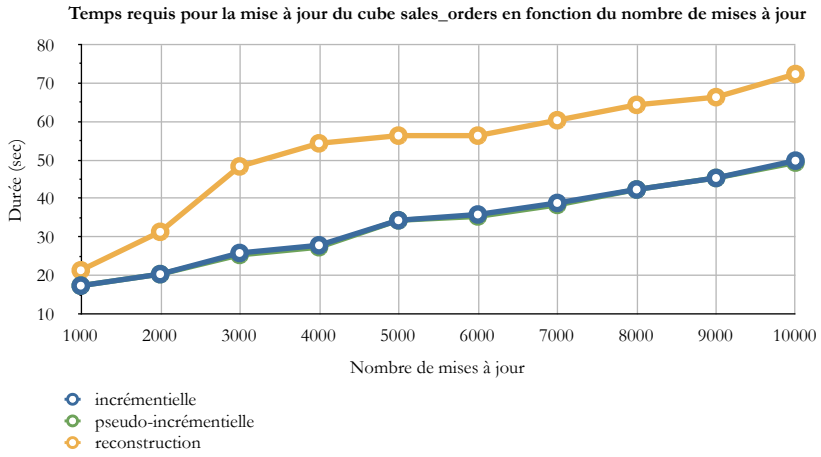


FIG. 5.14 – Temps d'exécution des procédures de mise à jour du cube *sales_orders*.

Cube *population_provinces* (Annexe C, Figures C.4 et C.6)

Le cube initial contient 52 416 enregistrements dans la table de faits, il comporte trois vues matérialisées. Nous avons testé la rapidité d'exécution de la mise à jour de cube pour des lots de mises à jour variant de 2 000 à 10 000 enregistrements, par incréments de 2 000, ensuite de 10 000 à 30 000 enregistrements par incréments de 10 000. Les résultats sont affichés en Figure 5.15.

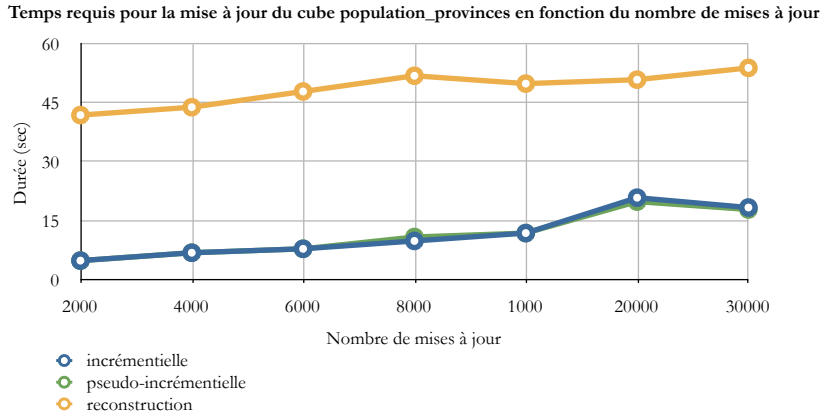


FIG. 5.15 – Temps d'exécution des procédures de mise à jour du cube *population_provinces*.

Les méthodes incrémentielle et pseudo-incrémentielle sont plus rapides que la reconstruction du cube (durée divisée par trois). La courbe pseudo-incrémentielle (en vert) est cachée par la courbe incrémentielle (en bleu), ce qui signifie que la reconstruction

des vues prend autant de temps que leur mise à jour incrémentielle, ceci du fait que la vue est formée par un regroupement (group by SQL) sur un faible volume de données.

Cube *population_divisions* (Annexe C, Figures C.3 et C.5)

Le cube initial contient 1 257 984 enregistrements dans la table de faits, il comporte trois vues matérialisées. Nous avons testé la rapidité d'exécution de la mise à jour de cube pour des lots de mises à jour variant de 1 000 à 25 000 enregistrements, par incréments de 5 000, ensuite pour 40 000 et 50 000 enregistrements et enfin de 100 000 à 400 000 enregistrements[15] par incréments de 100 000. Les résultats sont affichés en Figure 5.15.

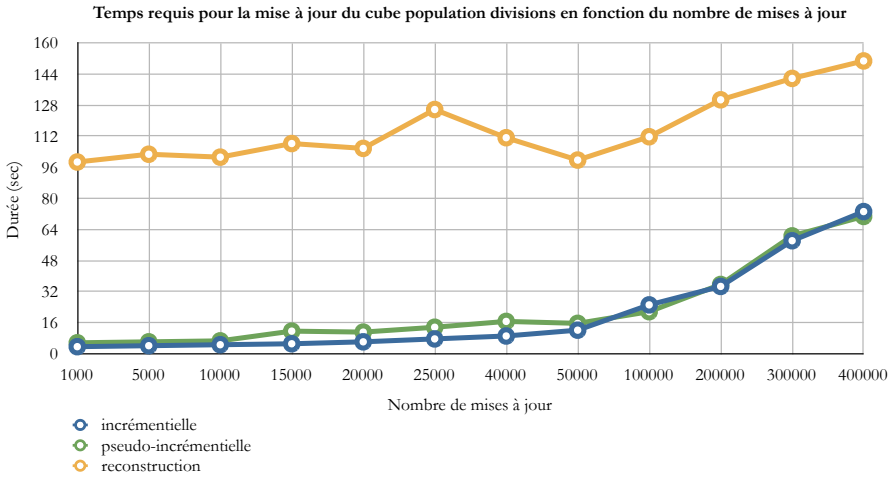


FIG. 5.16 – Temps d'exécution des procédures de mise à jour du cube *population_divisions*.

Les méthodes incrémentielle et pseudo-incrémentielle sont nettement plus rapides que la reconstruction du cube (rapport de 10 pour de petits volumes de mises à jour). Plus le volume de mises à jour est important, moins la méthode incrémentielle semble efficace. Il y aurait certainement un seuil où les trois courbes se rejoindraient et les méthodes mettront approximativement le même temps pour mettre à jour le cube, comme le suggèrent Ponniah [2001] avec le schéma présenté en Figure. 5.17. Par interpolation, pour nos procédures, ce seuil semblerait être atteint pour un volume de mises à jour

[15] 400 000 enregistrements représentent ici un tiers du cube.

de la moitié de la taille du cube. La méthode pseudo-incrémentielle n'est pas beaucoup plus coûteuse en temps que la méthode incrémentielle.

Interprétation générale des résultats obtenus

Les résultats obtenus sont satisfaisants, car les procédures incrémentielles et pseudo-incrémentielles sont plus rapides que la reconstruction totale du cube. Nos procédures de mise à jour de cube non spatial ont pu être validées : nous avons vérifié que les mises à jour étaient bien intégrées, et que les calculs des agrégations étaient exacts.

En résumé, pour un cube de très petite taille (moins de 30 000 enregistrements), la mise à jour incrémentielle s'avère peu avantageuse, car les procédures incrémentielles sont plus complexes à concevoir et ne réduisent pas énormément le temps requis pour la mise à jour. Pour des plus gros cubes, toujours de petite taille (entre 30 000 et 100 000 enregistrements), la mise à jour incrémentielle commence à être avantageuse et pour les cubes de taille moyenne (plus de 1 000 000 enregistrements), on note une nette différence de rapidité, la méthode incrémentielle est plus rapide pour les vues matérialisées et pour les faits lorsque le volume de mises à jour est faible par rapport à la taille du cube (moins de la moitié), mais devient plus coûteuse pour des mises à jour plus volumineuses.

Il est à noter que dans les entrepôts de données, les cubes sont souvent des cubes de plus grosses tailles que ceux avec lesquels nous avons effectué nos tests : les tables de faits contiennent souvent des dizaines ou centaines de millions d'enregistrements. Il nous a semblé futile de tester nos procédures sur de plus gros cubes, car si l'on considère deux cubes A et B de même structure, mais le cube B contient dix fois plus de faits que l'autre, on peut supposer que la procédure incrémentielle prendra autant de temps pour les cubes A et B, et que le temps nécessaire à la reconstruction totale du cube B serait proportionnel (d'un facteur 10) au temps de reconstruction du cube A. La différence réside donc uniquement dans le temps requis pour la procédure de reconstruction totale, et ce n'est pas celle qui nous intéresse ici. De plus, il est peu pratique d'effectuer des tests avec des cubes de si gros volumes, car la simulation des

mises à jour avec le programme Java sous Eclipse consomme beaucoup de temps (dizaines de minutes pour générer 1 000 000 de mises à jour). Pour ces raisons, nous n'avons pas souhaité tester nos procédures sur des cubes plus gros.
Dans chacun des cas, nous n'avons pas observé de nette différence entre les temps d'exécution des procédures pseudo-incrémentielles et incrémentielles, ceci peut s'expliquer par le fait que nos vues étaient d'assez petites tailles, et nous n'avons pas réellement pu observer les avantages d'utiliser les procédures de rafraîchissement incrémentiel des vues.

Ponniah [2001] proposent un graphique de comparaison du temps requis pour l'exécution de procédures de mise à jour de cube incrémentielles (update) avec la reconstruction totale (refresh) (cf. Fig. 5.17). Le temps d'exécution d'une procédure incrémentielle est presque proportionnel au volume de mises à jour (ajouts par incréments de même volume) alors que le temps d'exécution de la reconstruction des cubes est assez constant (peu croissant, car fonction du nombre d'enregistrements dans les tables du cube).

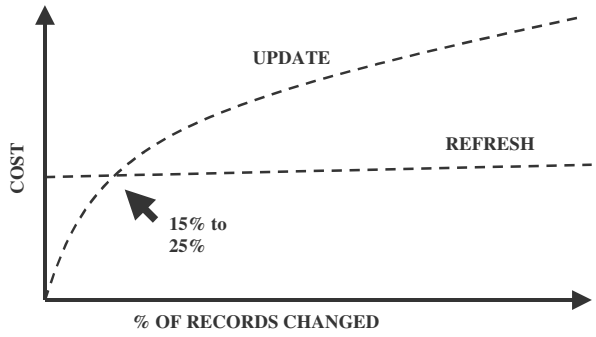


FIG. 5.17 – Comparaison du temps de mise à jour incrémentielle du cube avec le temps de reconstruction du cube [Ponniah, 2001]

Pour les deux cubes de population, nous nous plaçons dans la partie gauche du graphique Figure 5.17, nos résultats sont assez semblables. Par contre, pour le cube *sales_orders*, nos résultats diffèrent du modèle proposé. Ceci peut s'expliquer en raison de sa très petite taille, à ce niveau le temps requis est déterminé par l'enchaînement des procédures

au sein de l'ETL plutôt que par les procédures en elles-mêmes.

5.5.3 Tests des procédures de mise à jour de dimension spatiale

Nous avons testé la seconde solution proposée pour les procédures de mise à jour de la dimension spatiale *province* des deux cubes spatiaux, c.-à-d. la solution de mise à jour incrémentielle de la dimension spatiale proposée en 5.3.2. On a choisi de tester nos procédures ETL sur des géométries simples (polygones de moins de 20 points) et sur des géométries détaillées (polygones de plus de 100 points) pour observer l'influence de la géométrie sur la durée de la procédure. Les procédures sont exactement les mêmes pour les deux cubes ; nous pouvons alors parfaitement observer l'impact de la géométrie (détaillée ou moins détaillée) sur la durée de mise à jour de la dimension spatiale. Les comparaisons de performance sont établies sous forme de graphiques pour quatre scénarios de mise à jour :

1. incrémentielle avec extraction de toutes les données de la dimension contenues dans la source
2. incrémentielle avec extraction des données mises à jour uniquement
3. reconstruction totale de la dimension spatiale
4. reconstruction totale du cube

Les mise à jour sont saisies dans le logiciel SIG OpenSource uDig pour les deux jeux de données (géométrie simple et détaillée). La dimension spatiale d'origine comporte 12 membres (12 provinces). Pour chaque cube, nous avons testé la rapidité d'exécution pour 1, 4, 8 et 12 mises à jour (chaque mise à jour concerne une province différente).

Cube *population_provinces_spatial* (Annexe C, Figures C.7 et C.8)

Le cube *population_provinces_spatial* possède les mêmes dimensions et table de faits que le cube *population_provinces* à l'exception de la dimension spatiale *province*. Les géométries de la dimension spatiale sont très simplifiées, chaque membre est un polygone

qui comporte moins de 20 points. Les résultats sont affichés en Figure 5.18.

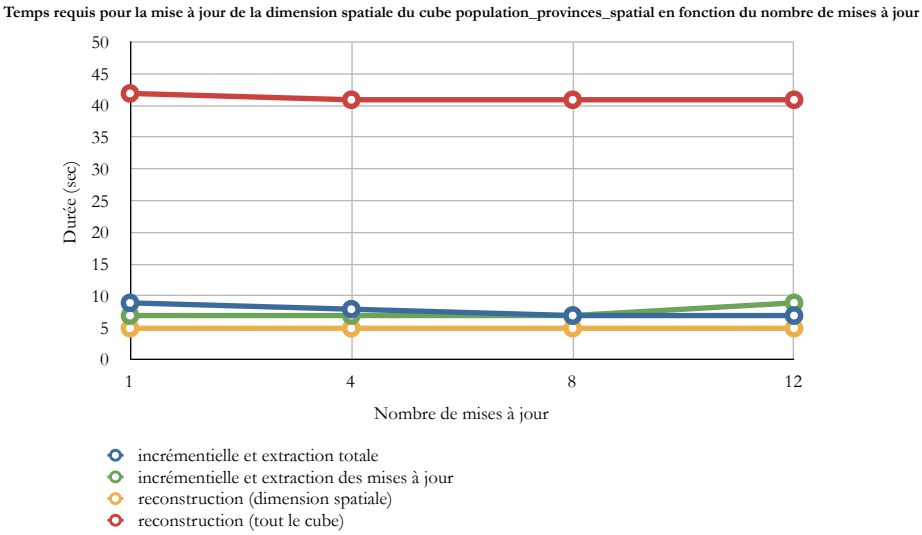


FIG. 5.18 – Durée d'exécution des procédures de mise à jour de la dimension spatiale du cube *population_provinces_spatial*.

La procédure de mise à jour incrémentielle est un peu plus lente que la reconstruction de la dimension. Les géométries étant simples, leur extraction prend peu de temps, mais d'une manière générale il est plus intéressant de n'extraire que les mises à jour, sauf si elles ont lieu sur tous les membres de la dimension spatiale (12 provinces ici). Dans nos scénarios de mise à jour qui mettent en jeu un faible volume de mises à jour (au plus 12 éléments, comme le montre le schéma de la Figure 5.19 où les mises à jour sont en rouge) , la reconstruction de la dimension est plus rapide, car elle fait uniquement appel aux procédures stockées du SGBD PostGIS servant à faire l'union de deux géométries et non à des programmes et librairies en Java comme GeOxygene ou Geotools. PostGIS est plus performant pour le calcul des géométries que ces dernières. La mise à jour incrémentielle est toutefois plus rapide que la reconstruction totale du cube.

Cube *population_provinces_spatial_detail* (Annexe C, Figures C.7 et C.8)

Le cube *population_provinces_spatial_detail* a la même structure que le cube précédent. Seules les données géométriques de la dimension spatiale sont différentes, la géométrie est beaucoup plus détaillée, les traitements spatiaux seront par conséquent plus longs.

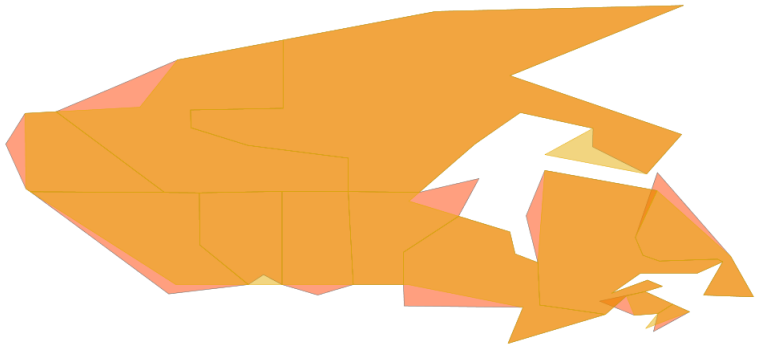


FIG. 5.19 – Mises à jour sur la dimension spatiale provinces avec géométries simplifiées.

Les résultats sont affichés en Figure 5.20.

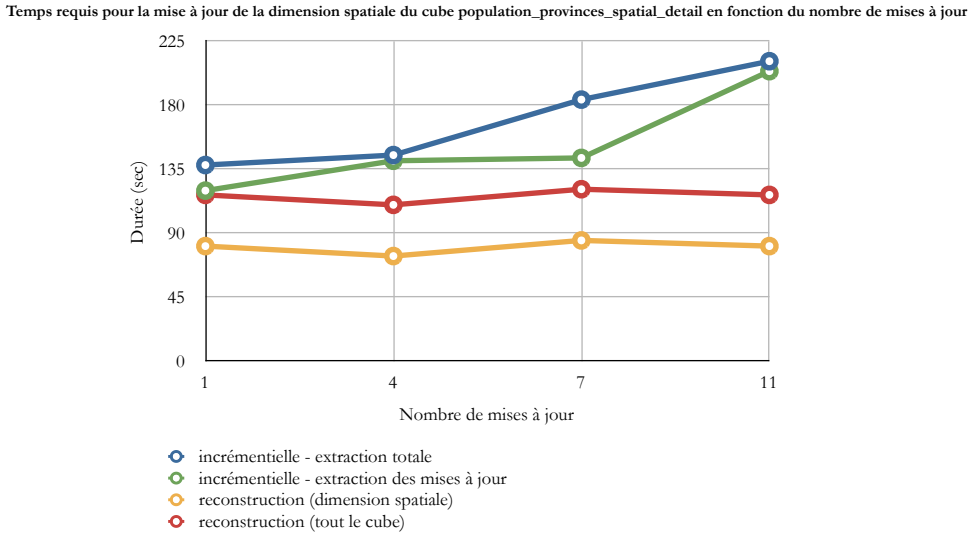


FIG. 5.20 – Durée d'exécution des procédures de mise à jour de la dimension spatiale du cube *population_provinces_spatial_detail*.

Pour les mêmes raisons que précédemment, la procédure de mise à jour incrémentielle est plus lente que la reconstruction de la dimension, et dépasse de beaucoup le temps requis pour la reconstruction de tout le cube.

De plus, les durées de mise à jour incrémentielle sont dix fois plus importantes que dans le cas précédent. La géométrie des éléments a donc beaucoup d'influence sur les procédures de mise à jour. La procédure incrémentielle avec extraction de toutes les données prend plus de temps que la procédure qui n'extrait que les mises à jour, en raison des coûts de lecture des géométries détaillées.

Interprétation générale des résultats obtenus pour la mise à jour de dimension spatiale

La première solution testée consistait en la propagation pseudo-incrémentielle des mises à jour, avec les vérifications nécessaires quant au respect des contraintes d'intégrité géométriques établies (solution proposée en 5.3.2). Nous avons rapidement arrêté les tests sur cette solution, car le temps requis pour les étapes de vérification de respect des contraintes géométriques devenait extrêmement long avec des géométries détaillées (plus de 10 minutes par membre et donc aurait pris des heures). La seconde solution, qui consiste en la propagation incrémentielle des mises à jour sans vérification des contraintes d'intégrité, apporte donc une amélioration considérable, mais reste moins rapide que la reconstruction de la dimension. Une des raisons à cela pourrait être le fait que la dimension est relativement petite, sa reconstruction est peu coûteuse, mais le deviendrait vite pour de plus grosses dimensions. D'une manière générale, le temps requis pour la mise à jour incrémentielle est croissant (avec le volume de mises à jour) tandis que le temps requis pour la reconstruction de la dimension spatiale est plus ou moins fixe (car fonction du nombre de membres de la dimension).

Pour des géométries simples, les procédures sont assez rapides, le temps de mise à jour de la dimension spatiale est beaucoup plus rapide que la mise à jour de tout le cube.

Lors des tests de rapidité sur des géométries détaillées, nos procédures de mise à jour incrémentielle de dimension spatiale furent plus lentes que la reconstruction totale du cube. Les raisons de ces moins bonnes performances peuvent être multiples et pour les découvrir, il serait nécessaire d'étudier plus amplement les différentes fonctions appelées lors de nos procédures de mise à jour et de comparer leur efficacité avec celles utilisées pour la reconstruction totale du cube. Puisque GeoKettle était encore en développement (une version alpha est sortie peu après nos travaux), il paraissait difficile de juger de ses performances.

En conclusion, une solution appropriée pour mettre à jour le cube serait d'opter pour des procédures de mise à jour incrémentielle pour les parties non spatiales et de reconstruire entièrement les dimensions spatiales à l'aide des opérateurs du SGBD.

5.6 Conclusion du chapitre

Ce chapitre décrit la mise en pratique des méthodes conceptuelles de mise à jour définies dans les chapitres 3 et 4. Le prototype de service web de mise à jour est conçu selon une architecture de composants distribués orientée services : les bases de données, l'outil ETL et le serveur web hébergeant le service. Le service web de mise à jour *SOAP_Geokettle_list* est destiné à l'administrateur du cube. Il offre des fonctionnalités de navigation au sein du référentiel ETL contenant les procédures, ainsi que la possibilité d'exécuter une procédure ETL de mise à jour.
La validation de notre prototype passe par la validation du **service web** et de nos **méthodes conceptuelles** de mise à jour. Le service web seul a été testé et fonctionne parfaitement, la partie la plus déterminante est la validation de nos méthodes.
Afin de valider les méthodes incrémentielles de mise à jour proposées dans les chapitres 3 et 4, nous les avons implantées sous forme de procédures dans l'outil ETL GeoKettle. Les tests effectués sont des comparaisons de la durée de la mise à jour en fonction du volume de mises à jour.

Les méthodes incrémentielles de **mise à jour de cube** ont été testées sur trois cubes non spatiaux de différentes tailles (1 000 à 1 200 000 enregistrements), elles comprennent les mises à jour tardives et récentes sur les dimensions et les tables de faits, ainsi que le rafraîchissement des vues matérialisées. La méthode incrémentielle a été comparée à la méthode pseudo-incrémentielle et à la reconstruction totale du cube. Les résultats obtenus sont très satisfaisants et concordent avec de précédents travaux de recherche [Ponniah, 2001]. Il est avantageux d'utiliser des procédures de mise à jour incrémentielle pour gérer un historique, dans le cas de gros cubes (plus de 1 000 000 enregistrements), pour un faible volume de mises à jour (moins du tiers du cube) ou dans le cas où l'on ne dispose que des mises à jour.

Les méthodes incrémentielles de **mise à jour de dimension spatiale** ont été testées sur deux cubes spatiaux, l'un contenant des géométries simples,

l'autre des géométries très détaillées. Le détail des éléments géométriques a une grande influence sur le temps d'exécution des procédures. Puisque la vérification des contraintes spatiales établies prenait trop de temps, nous avons opté pour une seconde solution effectuant moins d'opérations géométriques et répartissant l'exécution des opérations géométriques entre l'ETL et le SGBD source. Nos procédures de mise à jour incrémentielle de dimension spatiale sont plus coûteuses que la reconstruction totale de la dimension spatiale, et nous ne pouvons donc pas complètement valider nos méthodes de mise à jour de dimension spatiale sur le critère temporel. Il est toutefois avantageux d'utiliser la mise à jour incrémentielle pour garder un historique des modifications, ou bien si l'on ne dispose que des données mises à jour et qu'il nous est impossible de reconstruire le cube.

Nous avons traité tous les cas énoncés dans notre typologie de mise à jour de cube (cf. 2.2.3.2) et un seul cas de la typologie de mise à jour de dimension spatiale énoncée en 4.2.1. Les procédures implantées fonctionnent uniquement pour des cubes vérifiant les contraintes établies en 4.1.2.

La partie service web et les méthodes de mise à jour incrémentielles de cube du prototype sont donc validées. Les méthodes de mise à jour incrémentielle de dimension spatiale doivent être optimisées et il faudrait tester tous les cas de la typologie, ainsi que la mise à jour des mesures spatiales afin de fournir un prototype complet de service web de mise à jour de cube spatial.
En conclusion de nos recherches, la solution la plus rapide pour mettre à jour un cube spatial contenant des dimensions spatiales, mais aucune mesure spatiale, serait d'adopter une méthode de mise à jour hybride : mise à jour incrémentielle pour les parties non spatiales et reconstruction totale des dimensions spatiales.

Chapitre 6

Conclusions et perspectives

6.1 Conclusion

L'utilisation de méthodes de mise à jour incrémentielles des cubes de données se répand dans la sphère de l'informatique décisionnelle, mais la géomatique décisionnelle est un domaine récent et aucune méthodologie incrémentielle complète n'a été définie pour mettre à jour des cubes de données spatiales. La présente recherche est destinée à optimiser le processus de mise à jour des cubes spatiaux. Les résultats de nos travaux pourront être utiles à toutes les personnes en charge de la conception et de la mise à jour des cubes de données spatiales. Le travail effectué lors de cette maîtrise avait pour but premier de dégrossir le sujet en effectuant des lectures qui nous ont permis d'élaborer une typologie de mise à jour de cube. Le second objectif de la recherche était de proposer des méthodes de mise à jour incrémentielles pour des cubes de structure simple. Le dernier et troisième objectif de cette maîtrise était d'apporter une brique supplémentaire à la constitution d'une architecture orientée service pour la géomatique décisionnelle en concevant un service web de mise à jour de cube spatial.

Afin d'élaborer des méthodes de mise à jour incrémentielles pour les cubes spatiaux, nous avons tout d'abord étudié le sujet d'un point de vue plus large : la maintenance des entrepôts de données. Les recherches antérieures ont pointé du doigt la confusion régnant autour du terme « maintenance de l'entrepôt de données » et nous ont poussés

à élaborer une typologie pour cette maintenance, ce qui nous a permis de cadrer notre sujet de recherche et ensuite de formuler une typologie complète de la mise à jour de cube. Ainsi, par analogie avec le monde logiciel, nous avons défini les expressions « mise à jour de cube » et « mise à niveau de cube » : la mise à jour de cube ne doit avoir aucun impact sur les applications dérivées, elle touche le contenu du cube, tandis que la mise à niveau implique une adaptation des applications dépendantes (par exemple : serveurs OLAP, applications de Data Mining), elle concerne la structure du cube. Nous appuyant sur ces définitions, nous avons déterminé quatre types de mises à jour directes sur un cube de données : mises à jour récentes sur les dimensions, récentes sur la table de faits, tardives sur les dimensions et tardives sur la table de faits. Ces résultats sont le point central de notre recherche et serviront sans nul doute à de futurs travaux.

Nous avons ensuite étudié le processus et les méthodes existantes pour mettre à jour les cubes non spatiaux, en nous concentrant sur les méthodes incrémentielles. Le processus global et les méthodes concernant les quatre types de mise à jour énoncées ont été explicités.

Les cubes de données spatiales apportent de nouveaux concepts, dont les dimensions spatiales et les mesures spatiales. Il en ressort de nouvelles problématiques concernant leur mise à jour. Afin d'élaborer des méthodes nouvelles pour la mise à jour incrémentielle de cube spatial, nous avons détaillé et pris en compte les caractéristiques des cubes spatiaux. Nous nous sommes attachés à définir des méthodes incrémentielles pour la mise à jour des dimensions spatiales et des mesures spatiales géométriques. Nous avons proposé une typologie de mise à jour de la dimension spatiale et formulé des méthodes de mise à jour correspondantes à chaque type. La propagation des mises à jour au sein de la lattice de la dimension spatiale est traitée par analogie avec la propagation des mises à jour au sein de la lattice d'un cube non spatial. Pour la mise à jour des mesures spatiales numériques, nous avons adapté l'algorithme du domaine non spatial *Summary-delta* de rafraîchissement des vues matérialisées aux agrégats spatiaux. Le travail de recherche ne pouvait pas traiter tous les cas de cubes spatiaux possibles, aussi nous avons défini des contraintes d'intégrité, hypothèses restrictives qui nous ont

permis de travailler avec des cubes de structure simple [1].

Le prototype de service web de mise à jour de cube comprend deux ensembles : les procédures de mise à jour et le service web qui appelle ces procédures. La première étape dans l'élaboration de notre prototype concerna l'implantation des procédures de mise à jour de cube dans un outil ETL. Pour tester et valider nos procédures, nous avons créé nos propres jeux de données, car les données fournies par le projet GEOIDE ne contenaient pas de mises à jour. Le choix des logiciels (l'ETL, les SGBDs source et contenant le cube, ainsi que le serveur web) et du langage de programmation Java a été guidé par le souhait de bâtir notre prototype avec le plus de composants OpenSource possible, en vue de garantir une interopérabilité entre les systèmes, mais aussi par idéologie. Nous avons implanté deux types de processus de mise à jour dans l'outil ETL Geokettle : un processus de mise à jour de cube non spatial et un processus de mise à jour de dimension spatiale pour une modification de géométrie sur un membre détaillé de la dimension. Nous n'avons pas créé de processus de mise à jour de cube spatial complet, car l'implantation de toutes les méthodes de mise à jour des mesures spatiales dépassait le cadre de la maîtrise. Nous avons testé nos procédures de **mise à jour de cube non spatial** sur trois cubes, en comparant la durée de la mise à jour en fonction du volume de mises à jour, suivant trois scénarios : incrémentiel, pseudo-incrémentiel et reconstruction totale du cube. Les résultats sont satisfaisants et en accord avec de précédents travaux [Ponniah, 2001]. La méthode incrémentielle semble avantageuse dans la plupart des cas, car elle offre des durées d'exécution entre cinq et dix fois plus rapides que la reconstruction totale du cube, ce qui nous permet de valider nos procédures. Les procédures de **mise à jour de dimension spatiale** ont été testées suivant les mêmes critères sur deux cubes de même structure, mais possédant des géométries de niveaux de détail différents. Une première solution pseudo-incrémentielle avait été envisagée, mais abandonnée, car trop coûteuse en temps : aucune confiance n'était accordée quant à l'intégrité des jeux de données sources, on supposait que les mises à jour sur le niveau détaillé n'étaient pas forcément fournies intègres. Des vérifications de contraintes géométriques s'imposaient alors et

[1] Nous avons posé des contraintes explicitées au chapitre 5 afin de travailler avec des cas simplifiés.

ralentissaient considérablement le processus. Une seconde solution s'affranchissant des vérifications fut proposée, avec une méthode incrémentielle cette fois-ci, répartissant les opérations géométriques entre l'outil ETL et le SGBD source. Cette solution s'avère plus optimale, beaucoup moins coûteuse en temps d'exécution, mais reste toujours plus lente que la reconstruction totale de la dimension spatiale, ce qui ne nous permet pas de valider totalement notre approche. Les tests effectués ne permettent toutefois pas de tirer de conclusion sur la performance de nos méthodes, car nous avons testé les procédures sur de faibles volumes de mises à jour (une dizaine d'éléments au plus), il est fort à parier que les résultats seraient différents pour un plus gros volume de mises à jour, et que nos procédures pourraient s'avérer plus efficaces. Dans les deux solutions, nous avons remarqué que le détail des éléments géométriques avait une grande influence sur le temps d'exécution, la procédure est dix fois plus longue pour la dimension contenant des géométries détaillées.

La seconde étape dans l'élaboration de notre prototype fut donc l'élaboration du service web appelant ces procédures de mise à jour : nous souhaitions étudier la possibilité d'introduire la fonctionnalité de mise à jour de cube spatial au sein d'une architecture SOA. Pour cela nous avons exposé la mise à jour de cube sous forme de service web. Ce service web a permis d'élaborer une architecture orientée service autour des entrepôts de donnes spatiales. Cette architecture comprend cinq acteurs majeurs : le demandeur de service (administrateur du cube de données), le service web, le logiciel ETL avec les procédures, les sources de données et l'entrepôt de données contenant le cube. L'élaboration de notre service web s'inscrit donc dans les recherches effectuées par le groupe GeoSOA dirigé par le Dr. Thierry Badard et qui vise entre autres à concevoir une architecture orientées services pour faciliter la prise de décision en mobilité. Un service web de mise à jour a été conçu, implémenté et déployé au cours de la recherche. Il offre des fonctionnalités de recherche et d'exécution des procédures de mise à jour. Le service web a été validé avec succès : la structure des messages (de requête et de réponse) est conforme aux normes, le contenu des messages correspond à nos attentes, et les fonctions appelées sont bien exécutées.

En conclusion, puisqu'il n'existe encore aucun outil ETL optimal conçu pour les cubes de données spatiales, la solution la plus rapide pour intégrer un faible volume de mises à jour sur une dimension spatiale d'un cube pourrait s'avérer être une méthode hybride : incrémentielle pour les parties non spatiales et de reconstruire entièrement les dimensions spatiales. Aujourd'hui, nous pouvons regarder nos travaux avec plus de recul, nous sommes satisfaits de l'approche choisie et des résultats obtenus qui ont pu répondu à certaines questions (qu'est-ce que la mise à jour de cube ?) et soulevé d'autres problèmes (ne faut-il pas revoir la manière de stocker la géométrie dans les cubes de données pour une mise à jour plus rapide ?).

6.2 Perspectives

Les perspectives offertes par ce travail de recherche concernent principalement deux thématiques : l'architecture orientée service et la mise à jour des cubes de données spatiales.

Nous prétendons que l'évolution des systèmes d'information passera par le développement d'architectures orientées service, car elles présentent de nombreux avantages comme le respect des normes et l'intéropérabilité entre des composants répartis sur le réseau, qui sont accessibles à distance. L'OGC propose de nombreux standards pour l'échange et la gestion de l'information géographique. L'information spatiale est de plus en plus délocalisée, et il est nécessaire de la centraliser pour constituer des cubes de données alimentant un serveur de type SOLAP. Des standards comme WFS (Web Feature Service) permettant d'obtenir des objets spatiaux sous forme de GML pourraient être utilisés pour la livraison des mises à jour des sources vers le cube. L'outil ETL interrogerait le serveur de données sources qui lui enverrait les mises à jour dans un format XML. On pourrait également s'inspirer des travaux de Badard et Richard [2001] sur la transmission des mises à jour aux bases de données géographiques sous forme de XML. Il existe des ETL spatiaux pouvant lire et écrire beaucoup de formats, comme le logiciel FME de la société Safe Software, mais FME est orienté transactionnel et n'est pas conçu pour la gestion des données

multidimensionnelles. L'élaboration d'une telle architecture interopérable requerrait la conception d'un outil ETL spatial pour les données multidimensionnelles. La recherche menée au cours de la maîtrise ne s'est pas attachée à définir et implémenter une interface graphique pour faciliter l'appel des méthodes de mise à jour. La création d'une interface graphique pour la gestion du cube pourrait être réalisée lors de futurs travaux de recherche.

Les applications SOLAP sont vouées à évoluer vers le temps réel comme en témoignent de nombreux travaux de recherche [Lambert, 2006; Kuijpers et Vaisman, 2007; Raffaetà *et al.*, 2007]. Par exemple : la supervision du trafic, la gestion d'un réseau de téléphone mobile ou encore le suivi de marchandises en temps réel nécessitent des mises à jour rapides. On distinguerait deux types de dimensions spatiales : statiques, variant peu (limites administratives, routes, etc.) et dynamiques, variant continûment (piétons, autobus, etc.). Les dimensions dynamiques contiendraient donc des objets mobiles, et de nouvelles mesures seraient possibles : la présence, le nombre d'observations, la direction, la vitesse, etc. Les mesures spatiales sont encore peu utilisées dans les cubes spatiaux, la recherche s'est d'abord tournée vers l'implantation de dimensions spatiales, ceci étant fait, les recherches s'orientent désormais vers l'implantation de mesures spatiales dans le cube. Nous pensons que ces mesures sont promises à un bel avenir et qu'elles permettraient aux cubes d'offrir des applications dotées de capacités d'analyse spatio-temporelle inédites. Le présent travail de recherche a permis d'éclaircir le sujet de la mise à jour rapide des cubes spatiaux. La piste incrémentielle a été étudiée et testée : pour les cubes non spatiaux, nous avons testé les procédures de mise à jour incrémentielle sur les faits et les dimensions, pour les cubes spatiaux, nous avons testé la mise à jour de dimension spatiale uniquement (donc pas les mesures spatiales). La mise à jour incrémentielle des cubes de données spatiales est un sujet qui mérite d'être approfondi, notamment au niveau de la mise à jour des mesures spatiales. La définition d'une méthodologie complète pour la mise à jour incrémentielle de cube spatial, traitant différents types de cubes (avec plusieurs types de hiérarchies) et différents types de données spatiales (vectorielles : points, lignes et polygones, mais aussi des données raster), pourrait

faire l'objet d'une future thèse de doctorat. L'implantation de l'algorithme *Spatial Summary-delta* pourrait y être étudiée conjointement avec les recherches actuelles menées sur la définition d'opérateurs d'agrégation spatiale pour les mesures spatiales (thèse de Eve Grenier). Nous avons proposé deux solutions pour la mise à jour de la dimension spatiale statique, mais les résultats obtenus pour de faibles volumes de mises à jour ne sont pas entièrement satisfaisants. D'autres techniques mériteraient d'être étudiées, parmi elles, l'utilisation de la topologie pour stocker les géométries dans la dimension spatiale serait une alternative intéressante pour optimiser la mise à jour et le calcul des agrégations spatiales (p. ex. *Population des villes traversées par un cours d'eau*) [Gomez *et al.*, 2007]. Avec une topologie, la propagation des mises à jour aux niveaux supérieurs serait immédiate. Gérer toutes les géométries à l'aide d'une topologie est une manière de ne stocker qu'à un seul endroit les éléments, principe même de l'entrepôt de données et permettrait d'intégrer facilement tous les types de géométries : ponctuelle, linéaire et surfacique. Cette piste mérite d'être étudiée, bien que la topologie soit complexe à implémenter (coûteuse en espace mémoire, algorithmes plus complexes, besoin d'une base de données spatiales gérant la topologie, etc), elle pourrait être bénéfique et optimiser le processus de mise à jour des données spatiales au sein d'un cube de données. Si nous devions continuer un travail de doctorat sur le sujet, nous choisirions d'étudier cette piste pour stocker et mettre à jour la géométrie au sein des cubes de données spatiales (données de dimension et mesures).

Bibliographie

ADAMSON Christopher, Juin 2006 : *Mastering Data Warehouse Aggregates : Solutions for Star Schema Performance*. John Wiley & Sons.

BADARD Thierry, Décembre 2000 : *Propagation des mises à jour dans les bases de données géographiques multi-représentations par analyse des changements géographiques*. Thèse de doctorat, IGN.

BADARD Thierry et RICHARD Didier, 2001 : Using xml for the exchange of updating information between geographical information systems. *Computers, Environment and Urban Systems (CEUS), Elsevier Science Ltd*, 25:17–31.

BAKILLAH Mohammed, 2007 : Développement d'une approche géosémantique intégrée pour ajuster les résultats des requêtes spatio-temporelles dans les bases de données géospatiales multidimensionnelles évolutives. Mémoire de maîtrise, Université Laval.

BÉDARD Yvan, BODY Mathurin, MIQUEL Maryvonne et TCHOUNIKINE Anne, 2003 : Handling evolutions in multidimensional structures. *In Conference on Data Engineering (ICDE)*.

BÉDARD Yvan, MERRET Tim et HAN Jiawei, 2001 : *Fundamentals of spatial datawarehousing for geographic knowledge discovery*, volume Research Monographs in GIS de *Geographic Data Mining and Knowledge Discovery*, chapitre 3, pages 53–73. Taylor & Francis, première édition.

BÉDARD Yvan, MERRET Tim et HAN Jiawei, 2008 : *Fundamentals of spatial datawarehousing for geographic knowledge discovery*, chapitre 3. Geographic Data Mining and Knowledge Discovery. Taylor & Francis, seconde édition.

BÉDARD Yvan, RIVEST Sonia et PROULX Marie-Josée, 2006 : *Data Warehouses And Olap : Concepts, Architectures And Solutions*, chapitre 13. IRM Press.

BIMONTE Sandro, TCHOUNIKINE Anne et MIQUEL Maryvonne, 2006 : Geocube, a multidimensional model and navigation operators handling complex measures : Application in spatial olap. *In* ERICH NEUHOLD Tatyana Yakhno, éditeur : *Fourth Biennial International Conference on Advances in Information Systems*. Springer-Verlag.

BLAKELEY Jose A., LARSON Per-Ake et TOMPA Frank Wm., 1986 : Efficiently updating materialized views. *In SIGMOD Conference*, pages 61–71.

BLASCHKA Markus, SAPIA Carsten et HOFLING Gabriele, 1999 : On schema evolution in multidimensional databases. *In Data Warehousing and Knowledge Discovery*, pages 153–164.

BOUZEGHOUB Mokrane, FABRET Françoise et MATULOVIC-BROQUÉ Maja, 1999 : Modeling data warehouse refreshment process as a worklow application. *In Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW'99)*.

CARON P. Y, 1998 : L'étude du potentiel de olap pour supporter l'analyse spatio-temporelle. Mémoire de maîtrise, Université Laval.

CELLARY Wojciech et JOMIER Geneviève, 1990 : Consistency of versions in object-oriented databases. *In Proceedings of the 16th Very Large Databse Conference (VLDB'90), Brisbane, Australia*, pages 432–441. Morgan Kaufmann Publishers Inc. ISBN 1-55860-169-4.

CODD E.F., CODD S.b. et SALEY C.T., 1993 : Providing olap (on-line analytical processing) to user analyst : An it mandate. providing olap (on-line analytical processing) to user analyst : An it mandate.

DARMAWIKARTA Djoni, 2007 : *Dimensional Data Warehousing With MySQL*. Brainysoftware.

DUBÉ Etienne, BADARD Thierry et BÉDARD Yvan, Septembre 2007a : Building geospatial business intelligence solutions with free and open source components. FOSS4G.

DUBÉ Etienne, BADARD Thierry et BÉDARD Yvan, 2007b : Service Web de constitution en temps réel de mini-cubes SOLAP pour clients mobiles : une architecture orientée services pour l'utilisation mobile des données géo-décisionnelles. *In SAGEO 2007, Rencontres internationales Géomatique et territoire (CD-ROM)*, Clermont-Ferrand, France, 2007b. ISBN 978-2-85710-078-2.

DYRESON Curtis, GRANDI Fabio, KäFER Wolfgang, KLINE Nick, LORENTZOS Nikos, MITSOPOULOS Yannis, MONTANARI Angelo, NONEN Daniel, PERESSI Elisa, PERNICI Barbara, RODDICK John F., SARDA Nandlal L., SCALAS Maria Rita, SEGEV Arie, SNODGRASS Richard Thomas, SOO Mike D., TANSEL Abdullah, TIBERIO Paolo et WIEDERHOLD Gio, 1994 : A consensus glossary of temporal database concepts. *SIGMOD Rec.*, 23(1):52–64. ISSN 0163-5808.

EDER Johann et KONCILIA Christian, 2003 : Temporal versioning in data warehouses. pages 73–97.

FAVRE C, BENTAYEB F et BOUSSAID O, 2007 : *A Survey of Data Warehouse Model Evolution*. Encyclopedia of Database Technologies and Applications. Idea Group Publishing, seconde édition.

FERRI Fernando, POURABBAS Elaheh, RAFANELLI Maurizio et RICCI F. L., 2000 : Extending geographic databases for a query language to support queries involving statistical data. *In SSDBM '00 : Proceedings of the 12th International Conference on Scientific and Statistical Database Management (SSDBM'00)*, page 220, Washington, DC, USA, 2000. IEEE Computer Society. ISBN 0-7695-0686-0.

FOERSTER Theodor, 2006 : An open software framework for web service-based geoprocesses. *In Free and Open Source Software for Geoinformatics*, Lausanne, Switzerland, 2006.

FRANKLIN Carl, 1992 : An introduction to geographic information systems : linking maps to databases. *Database*, 15(2):12–21. ISSN 0162-4105.

FRIGIONI Daniele et TARANTINO Laura, 2003 : Multiple zooming in geographic maps. *Data Knowl. Eng.*, 47(2):207–236. ISSN 0169-023X.

Gomez L., Haesevoets S., Kuijpers B. et Vaisman A., 2007 : Spatial Aggregation : Data Model and Implementation. *ArXiv e-prints*, 707.

GRAY Jim, CHAUDHURI Surajit, BOSWORTH Adam, LAYMAN Andrew, REICHART Don et VENKATRAO Murali, 1997 : Data cube : A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–53.

GRENIER Eve, Avril 2007 : Élaboration d'une approche d'agrégation de données géospatiales pour le peuplement de cubes géospatiaux. Examen pré-doctoral.

GRIFFIN Timothy et LIBKIN Leonid, 1995 : Incremental maintenance of views with duplicates. *In SIGMOD '95 : Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 328–339, New York, NY, USA, 1995. ACM. ISBN 0-89791-731-6.

GUPTA Ashish et MUMICK Iderpal Singh, 1999a : *Materialized views : techniques, implementations, and applications*. MIT Press, Cambridge, MA, USA. ISBN 0-262-57122-6.

GUPTA Ashish et MUMICK Inderpal Singh, 1999b : Maintenance policies. *Materialized views : techniques, implementations, and applications*, pages 9–11.

GUPTA Ashish, MUMICK Inderpal Singh et SUBRAHMANIAN V. S., 1993 : Maintaining views incrementally. *In SIGMOD '93 : Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 157–166, New York, NY, USA, 1993. ACM. ISBN 0-89791-592-5.

HARINARAYAN Venky, RAJARAMAN Anand et ULLMAN Jeffrey D., 1996 : Implementing data cubes efficiently. *SIGMOD Rec.*, 25(2):205–216. ISSN 0163-5808.

HAUNERT Jan-Henrik et WOLFF Alexander, 2006 : Generalization of land cover maps by mixed integer programming. *In GIS '06 : Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pages 75–82, New York, NY, USA, 2006. ACM. ISBN 1-59593-529-0.

HORNER John, SONG Il-Yeol et CHEN Peter P., 2004 : An analysis of additivity in olap systems. *In DOLAP '04 : Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, pages 83–91, New York, NY, USA, 2004. ACM. ISBN 1-58113-977-2.

HUMPHRIES Mark, W.HAWKINS Michael et C.DY Michelle, Décembre 1998 : *Data Warehousing : Architecture and Implementation.* Prentice Hall, première édition.

IMHOFF Claudia, GALEMMO Nicholas et GEIGER Jonathan G., Août 2003 : *Mastering Data Warehouse Design : Relational and Dimensional Techniques.* Wiley.

INMON W. H., 2005 : *Building the Data Warehouse, 4th Edition.* Wiley, quatrième édition.

JEANSOULIN Robert, MOLENAAR Martien, WORBOYS Michael, FISHER Peter F., FRANK Andrew U. et MONGÉNIE Pierre, 2000 : Revigis : Uncertain knowledge maintenance and revision in geographic information systems. Rapport technique, Information Societies Technology, European Union.

KENNEDY Dennis, Avril 2003 : La réalité de olap temps réel. Microsoft Corporation.

KIMBALL Ralph, Novembre 1995 : The aggregate navigator. *DBMS online.*

KIMBALL Ralph et CASERTA Joe, Septembre 2004 : *The Data Warehouse ETL Toolkit : Practical Techniques for Extracting, Cleaning, Conforming and Delivering Data.* Wiley.

KIMBALL Ralph et ROSS Margy, Avril 2002 : *The Data Warehouse Toolkit : The Complete Guide to Dimensional Modeling (Second Edition).* Wiley. ISBN 0471200247.

KUIJPERS Bart et VAISMAN Alejandro A., 2007 : A data model for moving objects supporting aggregation. *In Proceedings of the ICDE Workshop on Spatio-Temporal Data Mining.*

LAMBERT Mélanie, 2006 : Développement d'une approche pour l'analyse solap en temps réel : adaptation aux besoins des activités sportives en plein air. Mémoire de maîtrise, Université Laval, Québec.

LARMAN Craig, 2004 : *Agile and Iterative Development : A Manager's Guide.* Addison-Wesley.

LEVESQUE Marie-Andrée, BÉDARD Yvan, GERVAIS Marc et DEVILLERS Rodolphe, Juin 2007 : Towards managing the risks of data misuse for spatial datacubes. *In Proceedings of the 5th International Symposium on Spatial Data Quality.*

MACKANESS W.A. et CHAUDHRY O., 2008 : *International Encyclopedia of Human Geography*, chapitre Cartographic Generalisation. R. Kitchin and Nigel Thrift.

MALINOWSKI Elzbieta et ZIMáNYI Esteban, 2004 : Representing spatiality in a conceptual multidimensional model. *In GIS '04 : Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 12–22, New York, NY, USA, 2004. ACM Press. ISBN 1-58113-979-9.

MALINOWSKI Elzbieta et ZIMáNYI Esteban, Estebannyi, 2005 : Spatial hierarchies and topological relationships in the spatial multidimer model. *Database : Enterprise, Skills and Innovation*, pages 17–28.

MARCHAND Pierre, BRISEBOIS Alexandre, EDWARDS Geoffrey et BÉDARD Yvan, 2004 : Implementation and evaluation of a hypercube-based method for spatio- temporal exploration and analysis. *ISPRS Journal of Photogrammetry & Remote Sensing*, 59(1-2):6–20.

MCHUGH Rosemary, 2008 : Etude du potentiel de la structure matricielle pour optimizer l'analyse spatiale de données géodécisionnelles. Mémoire de maîtrise, Université Laval.

MUMICK Inderpal Singh, QUASS Dallan et MUMICK Barinderpal Singh, 1997 : Maintenance of data cubes and summary tables in a warehouse. *SIGMOD Rec.*, 26(2):100–111. ISSN 0163-5808.

NAKOS B., S. S. Mustière et GAFFURI J., 2008 : A transition from simplification to generalisation of natural occuring lines. Rapport technique.

NEWCOMER Eric, 2002 : *Understanding Web Services*. Addison-Wesley Professional.

ORACLE, Mars 2002 : *Oracle9i Data Warehousing Guide*. Oracle.

PAPADIAS Dimitri, TAO Yufei, KALNIS Panos et ZHANG Jun, 2002 : Indexing spatio-temporal data warehouses. *In ICDE '02 : Proceedings of the 18th International Conference on Data Engineering*, page 166, Washington, DC, USA, 2002. IEEE Computer Society.

PAPADIAS Dimitris, KALNIS Panos, ZHANG Jun et TAO Yufei, 2001 : Efficient OLAP operations in spatial data warehouses. *Lecture Notes in Computer Science*, 2121:443–??

PEDERSEN Torben Bach et TRYFONA Nectaria, 2001 : Pre-aggregation in spatial data warehouses. *In SSTD '01 : Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, pages 460–480, London, UK, 2001. Springer-Verlag. ISBN 3-540-42301-X.

PENDSE Nigel, Février 2005a : Database explosion. The OLAP Report.

PENDSE Nigel, Février 2005b : What is olap ? The OLAP Report.

PONNIAH Paulraj, Août 2001 : *Data Warehousing Fundamentals : A Comprehensive Guide for IT Professionals*. Wiley-Interscience, première édition.

POULIOT Jacynthe, BÉDARD Yvan, CARON Claude, LEVASSEUR Claude, MÉTIVIER Romain et MONAGHAN Dave, 2004 : M@jic : Expérimentation d'une approche incrémentielle de gestion et d'échange de mises à jour de données géospatiales. *GEOMATICA*, 58(2):119–132.

RAFANELLI Mauizio, march 2003 : *Multidimensional Databases : Problems and Solutions*. IGI Global.

RAFFAETÀ A., BRAZ F., ORLANDO S., ORSINI R., RONCATO A. et SILVESTRI C., 2007 : Approximate aggregations in trajectory data warehouses. *In Proceedings of the ICDE Workshop on Spatio-Temporal Data Mining*.

RAGEUL Nicolas, 2007 : Vers une optimisation du processus d'analyse en ligne des données 3d : cas des fouilles archéologiques. Mémoire de maîtrise, Université Laval.

RIVEST Sonia, BÉDARD Yvan et MARCHAND Pierre, 2001 : Toward better support for spatial decision making : defining the characteristics of spatial on-line analytical processing. *Geomatica*, 55(4):539–555.

RIVEST Sonia, BÉDARD Yvan, PROULX Marie-Josée, NADEAU Martin, HUBERT Frederic et PASTOR Julien, 2005a : Solap technology : Merging business intelligence with geospatial technology for interactive spatio-temporal exploration and analysis of data. *ISPRS Journal of Photogrammetry & Remote Sensing*.

RIVEST Sonia, BEDARD Yvan, PROULX Marie-Josee, NADEAU Martin, HUBERT Frederic et PASTOR Julien, 2005b : Solap technology : Merging business intelligence with geospatial technology for interactive spatio-temporal exploration and analysis of data. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(1):17–33.

SABO M. et BÉDARD Y., Octobre 2007 : Enrichissement des bases de données spatiales afin de supporter un processus de généralisation cartographique à la volée. *In Géocongrès Québec 2007*.

SALEHI M., BÉDARD Y., MOSTAFAVI M. A. et BRODEUR J., 2007 : From transactional spatial databases integrity constraints to spatial datacubes integrity constraints. *In Spatial Data Quality 2007 (ISSDQ07)*.

SCALZO Bert, Juillet 2003 : *Oracle DBA Guide to Data Warehousing and Star Schemas*. Prentice Hall PTR, première édition.

SHMUELI Oded et ITAI Alon, 1984 : Maintenance of views. *In SIGMOD '84 : Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 240–255, New York, NY, USA, 1984. ACM. ISBN 0-89791-128-8.

SHUKLA Amit, DESHPANDE Prasad, NAUGHTON Jeffrey F. et RAMASAMY Karthikeyan, 1996 : Storage estimation for multidimensional aggregates in the presence of hierarchies. *In The VLDB Journal*, pages 522–531.

STEFANOVIC Nebojsa, HAN Jiawei et KOPERSKI Krzysztof, 2000 : Object-based selective materialization for efficient implementation of spatial data cubes. *IEEE Transactions on Knowledge and Data Engineering*, 12(6):938–958. ISSN 1041-4347.

THIERNEY Janet, Mars 2006 : Sas cube update, new functionnality. *In SAS users group international (San Francisco)*. SAS OLAP Server R&D.

TØSSEBRO Erlend et GÜTING Ralf Hartmut : Creating representations for continuously moving regions from observations.

VAISMAN Alejandro A., 2001 : *Updates, View Maintenance and Time Management in Multidimensional Databases*. Thèse de doctorat, Universidad de Buenos Aires.

WORBOYS M., 1998 : Modelling changes and events in dynamic spatial systems with reference to socio-economic units.

Annexe A

Fichier XML de définition d'un schéma pour Mondrian[1]

```
<?xml version="1.0" encoding="UTF-8"?>
<Schema>
<Cube name="Sales">
<Table name="sales_fact_1997"/>
<Dimension name="Gender" foreignKey="customer_id">
<Hierarchy hasAll="true" allMemberName="All_Genders" primaryKey="customer_id">
<Table name="customer"/>
<Level name="Gender" column="gender" uniqueMembers="true"/>
</Hierarchy>
</Dimension>
<Dimension name="Time" foreignKey="time_id">
<Hierarchy hasAll="false" primaryKey="time_id">
<Table name="time_by_day"/>
<Level name="Year" column="the_year" type="Numeric" uniqueMembers="true"/>
<Level name="Quarter" column="quarter" uniqueMembers="false"/>
<Level name="Month" column="month_of_year" type="Numeric" uniqueMembers="false"/
    >
</Hierarchy>
</Dimension>
<Measure name="Unit_Sales" column="unit_sales" aggregator="sum" formatString="
    #,###"/>
<Measure name="Store_Sales" column="store_sales" aggregator="sum" formatString="
    #,###.##"/>
<Measure name="Store_Cost" column="store_cost" aggregator="sum" formatString="
    #,###.00"/>
<CalculatedMember name="Profit" dimension="Measures" formula="[Measures].
_[Store_Sales]-[Measures].[Store_Cost]">
<CalculatedMemberProperty name="FORMAT_STRING" value="$#,##0.00"/>
</CalculatedMember>
</Cube>
</Schema>
```

[1] Schéma fourni sur le site de la société : http ://www.pentaho.org

Annexe B

Exemple de requête MDX invalidée par une mise à jour de cube

Prenons l'exemple d'un cube contenant trois dimensions : sexe, âge et localisation. La requête MDX[1] suivante demande population de la ville de Montréal :

```
SELECT {[Mesures].[Population]} ON COLUMNS
FROM Population WHERE
[Pays].[Canada].[Quebec].[Montreal]
```

Cette requête risque de ne plus être valable si une mise à jour indique que Montréal appartient désormais à l'Ontario, puisqu'il faudrait alors écrire :

```
SELECT {[Mesures].[Population]} ON COLUMNS
FROM Population WHERE
[Pays].[Canada].[Ontario].[Montreal]
```

[1] Inventé par Mosha Pasumansky au sein de Microsoft, MDX (Multidimensional Expressions) est un langage de requête pour les bases de données OLAP.

Annexe C

Schémas conceptuels des cubes de données

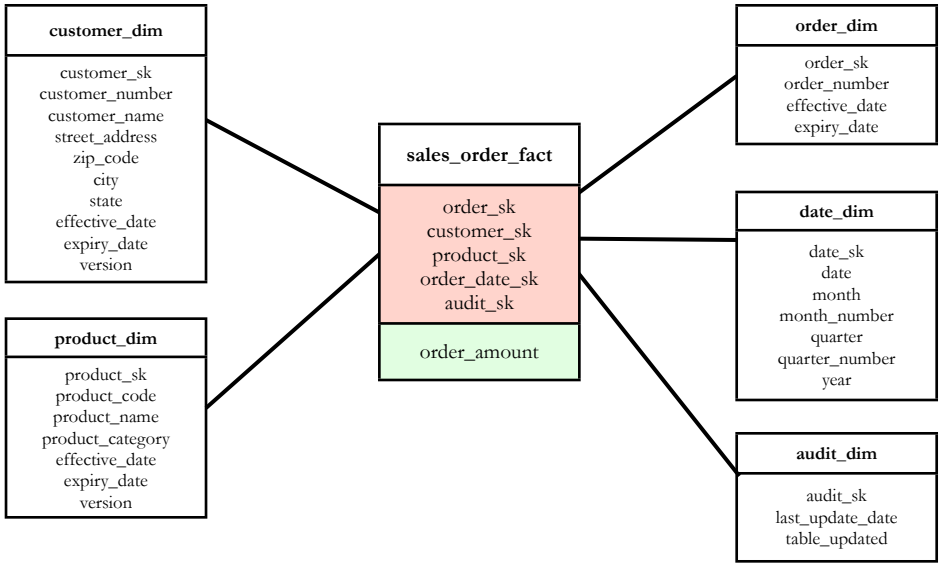


FIG. C.1 – Schéma conceptuel du cube non spatial *sales_order*.

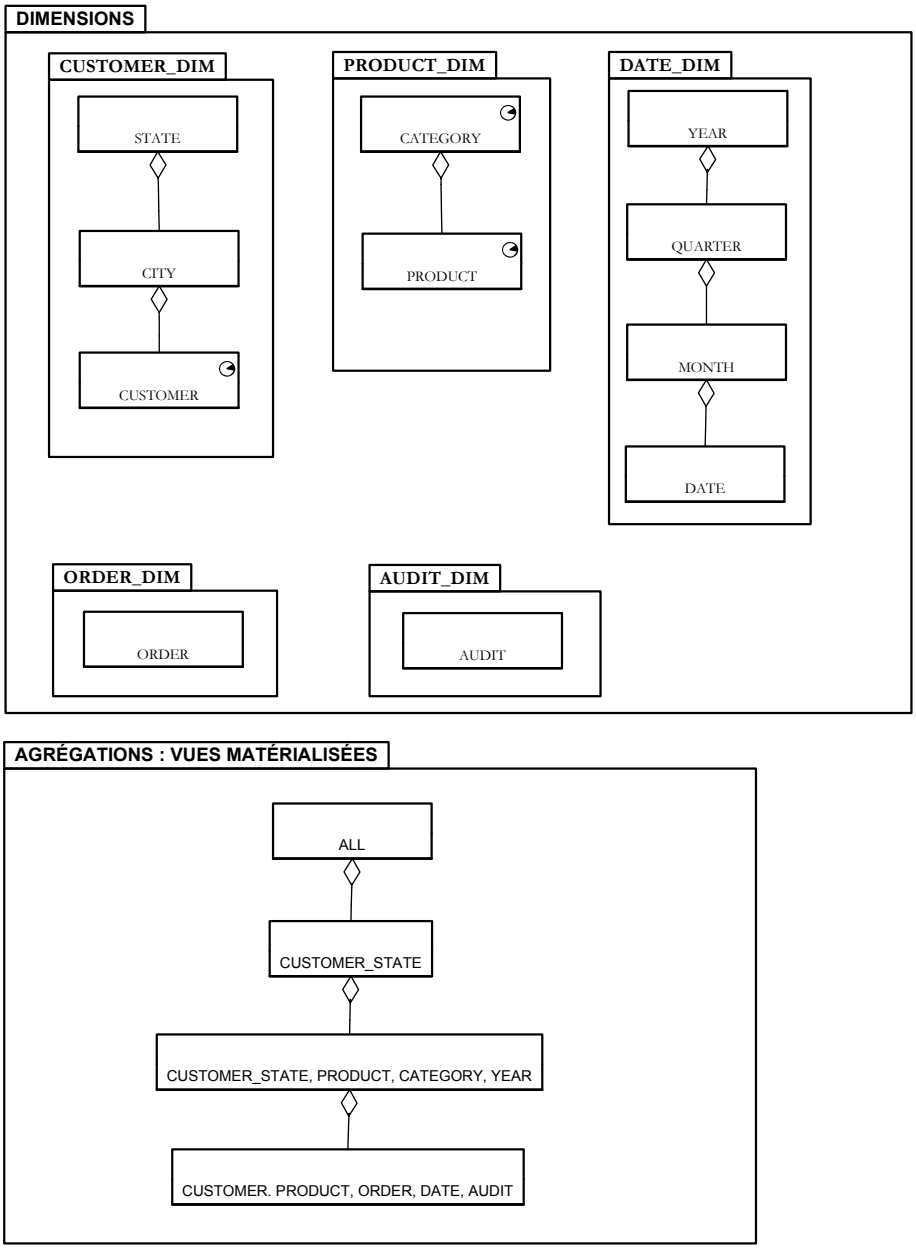


FIG. C.2 – Lattices des dimensions et des vues du cube non spatial *sales_order.*

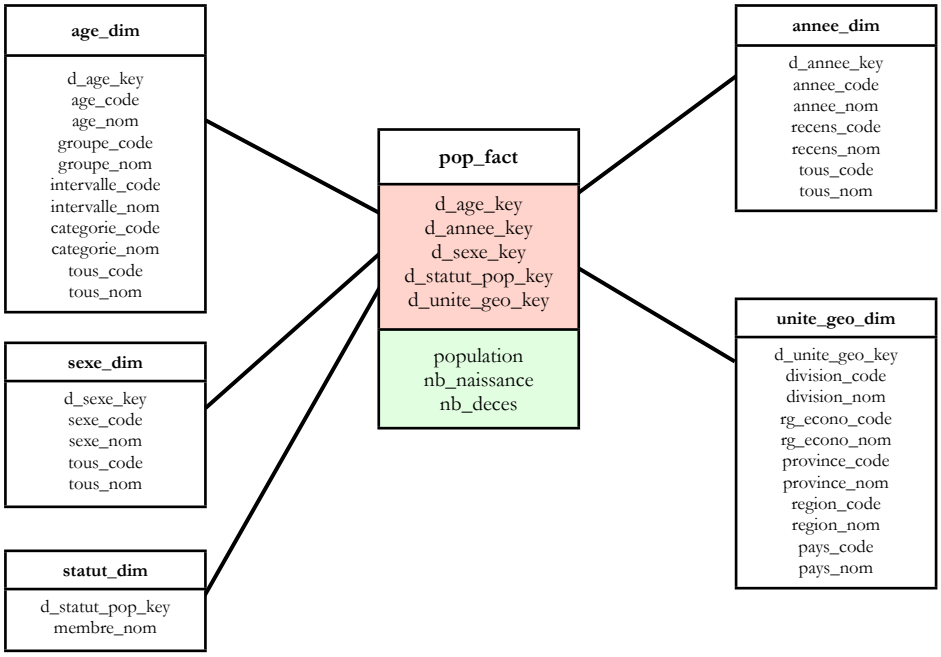


FIG. C.3 – Schéma conceptuel du cube non spatial *population_divisions.*

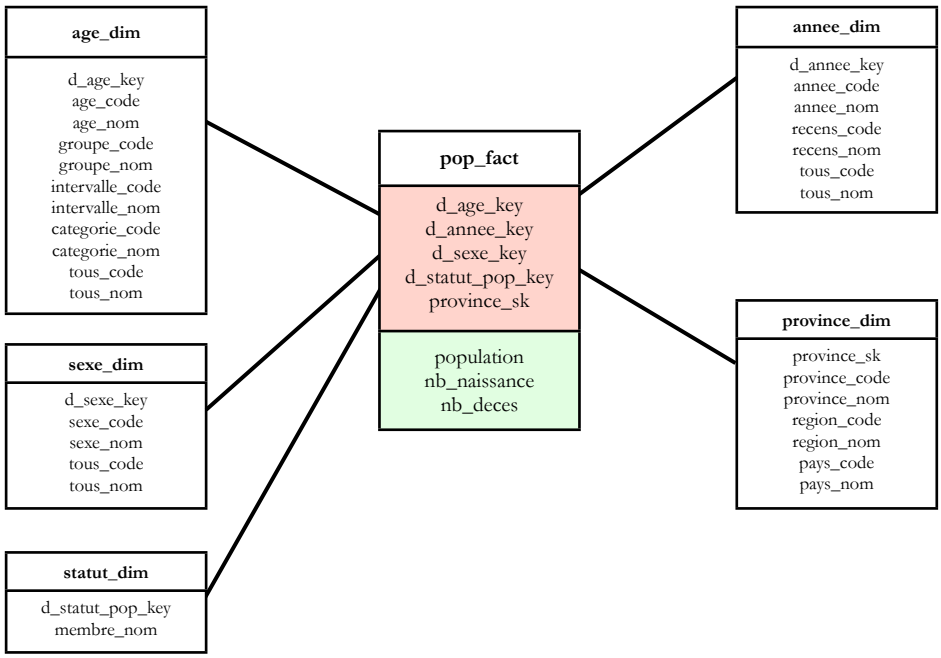


FIG. C.4 – Schéma conceptuel du cube non spatial *population_provinces.*

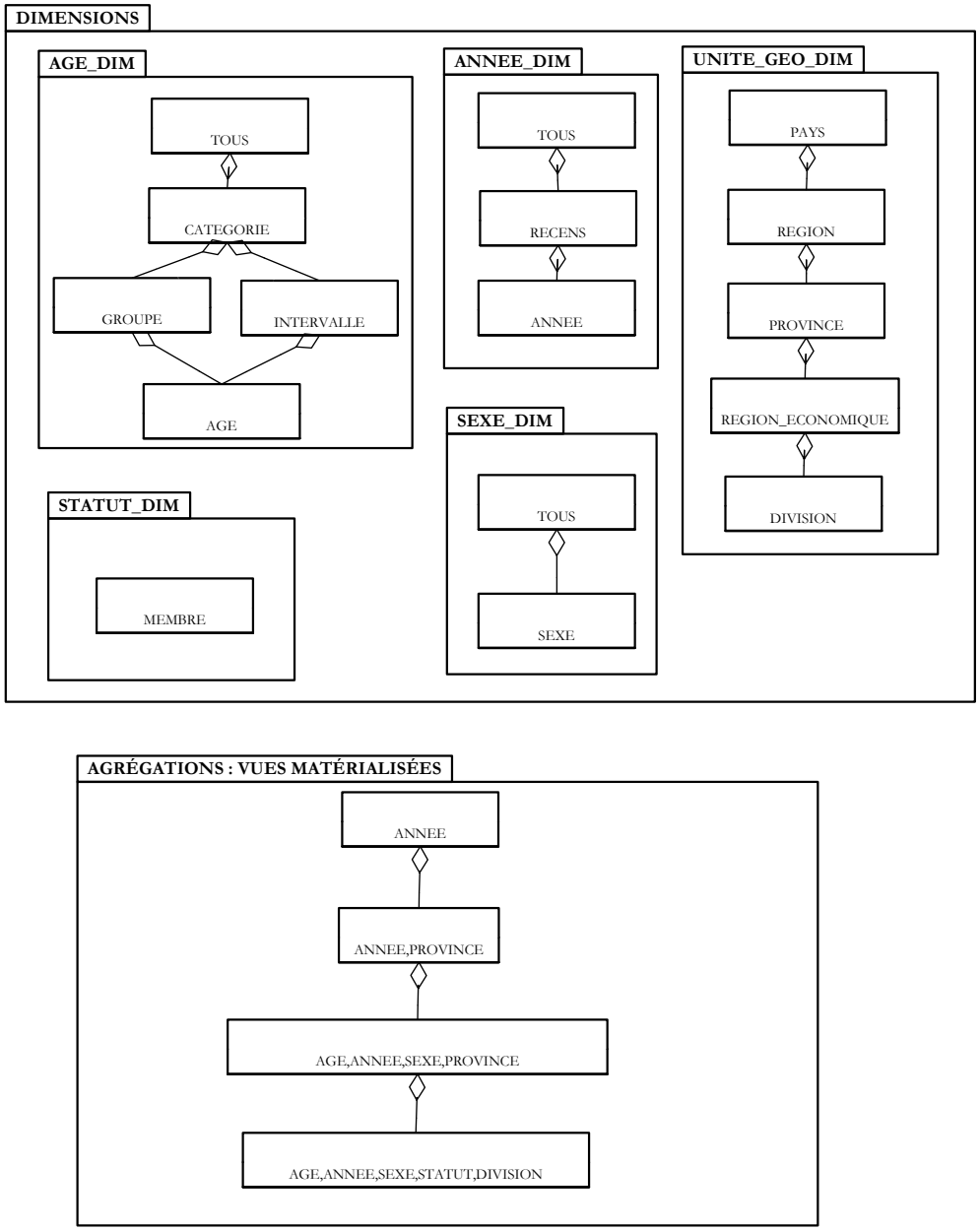


FIG. C.5 – Lattices des dimensions et des vues du cube non spatial *population_divisions.*

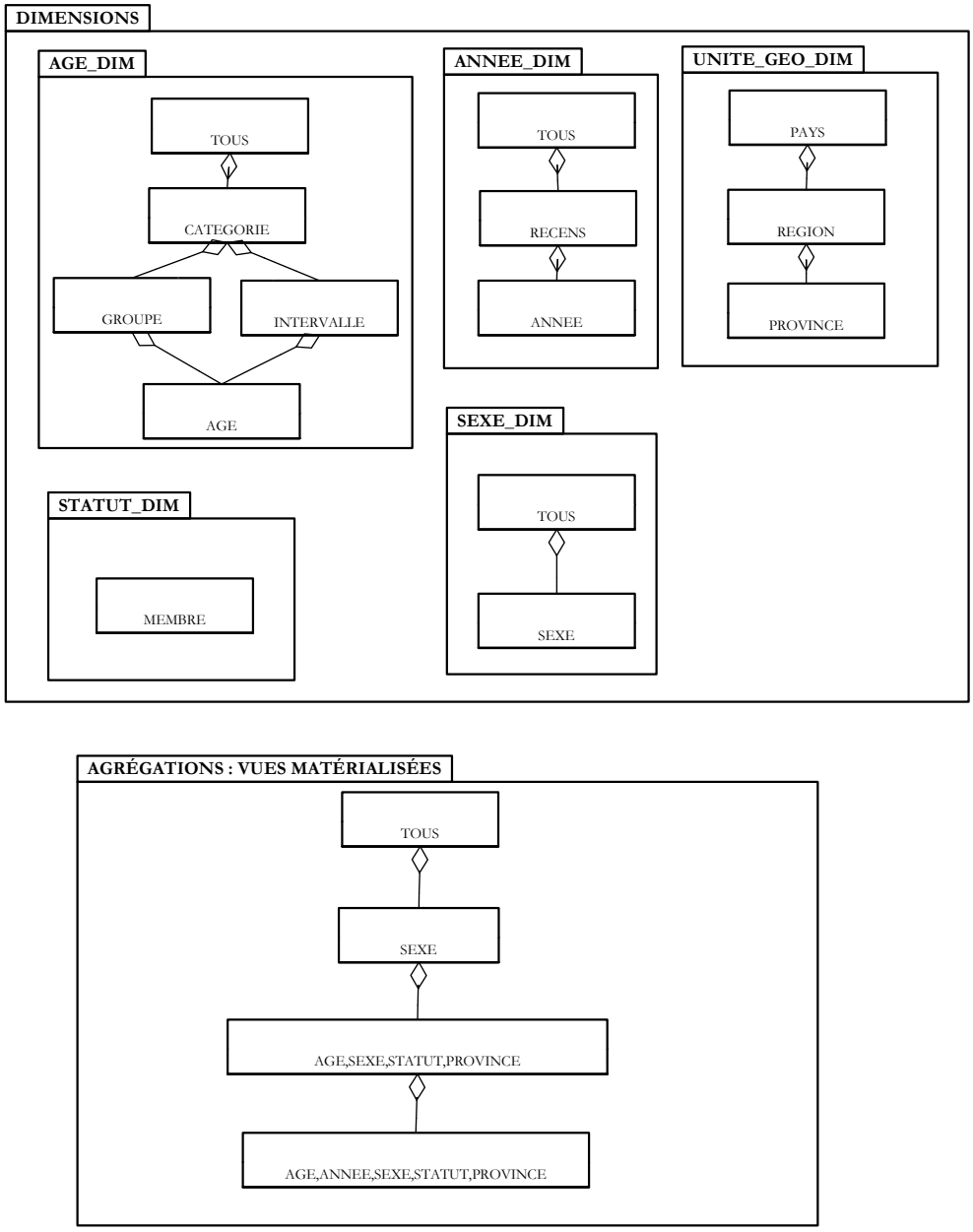


FIG. C.6 – Lattices des dimensions et des vues du cube non spatial *population_provinces*.

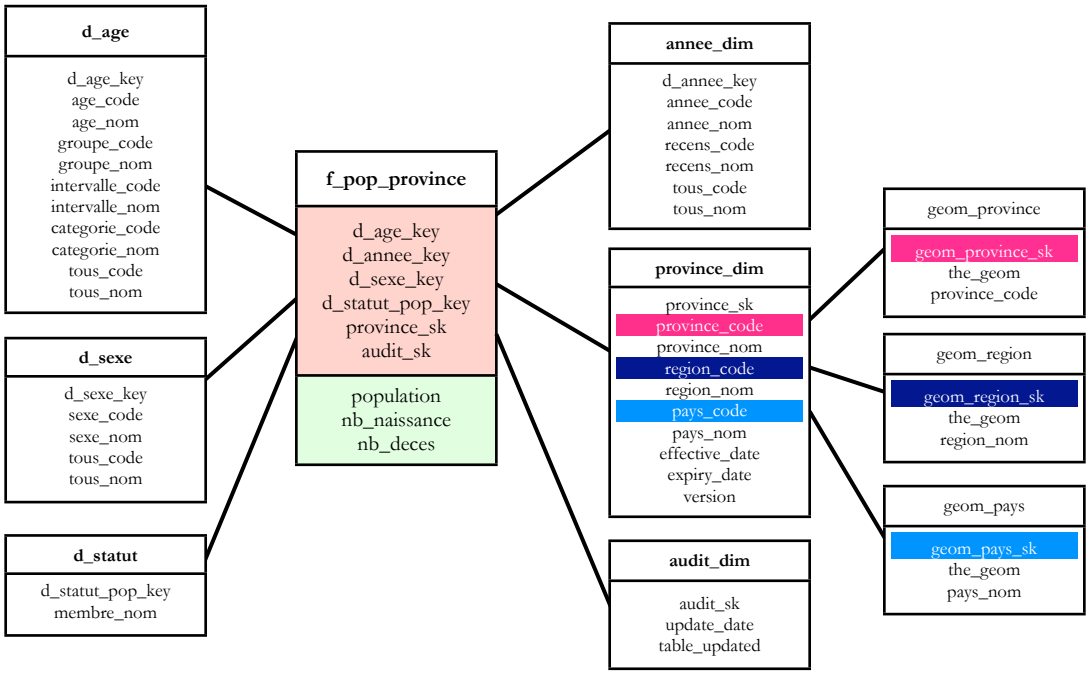


FIG. C.7 – Schéma conceptuel du cube spatial *population_provinces*.

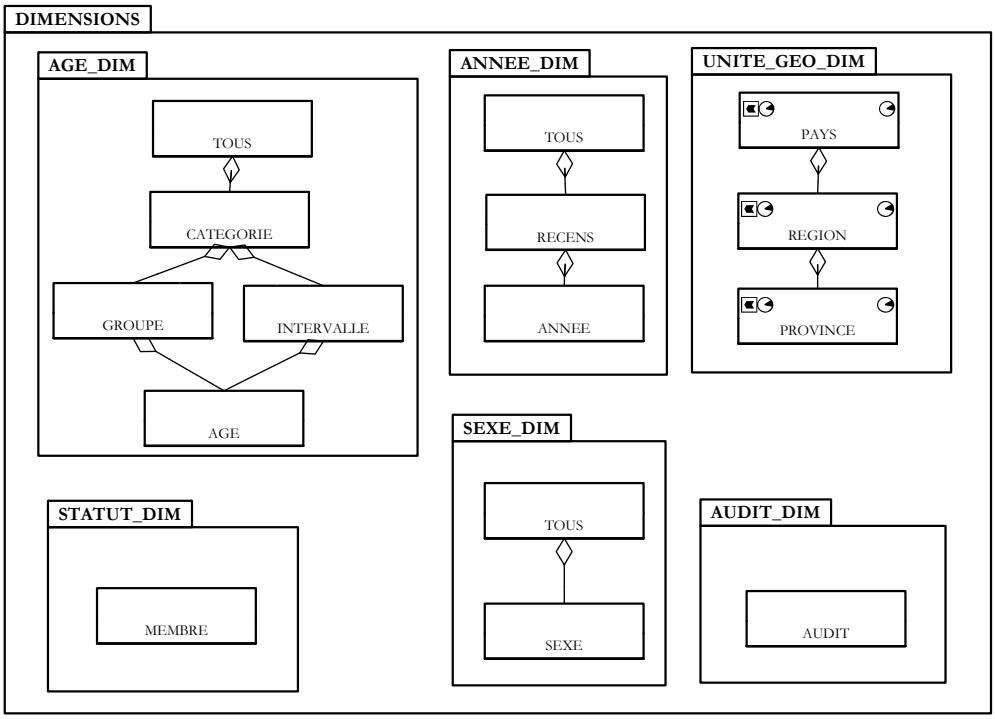


FIG. C.8 – Lattices des dimensions et des vues du cube spatial *population_provinces*.

Annexe D

Fichiers WSDL et SOAP du service web

Les fichiers XML sont tirés du service web *SOAP_Geokettle_list* (style Remote Procedure Call ou RPC) qui met à disposition les opération suivantes :

- listrep : lister les référentiels
- listdir : lister les répertoires
- listjobs : lister les jobs
- execute : exécuter un job

D.1 Exemple de fichier WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="http://scg.ulaval.ca/SOAP_Geokettle_list/"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="SOAP_Geokettle_list"
    targetNamespace="http://scg.ulaval.ca/SOAP_Geokettle_list/">

    <!-définition des éléments échangés-->
    <!--aucune car style RPC-->

    <!-description des messages échangés pour chaque opération-->
    <wsdl:message name="listrepRequest">
        <wsdl:part name="listrepRequest" type="xsd:string" />
    </wsdl:message>
    <wsdl:message name="listrepResponse">
        <wsdl:part name="listrepResponse" type="xsd:string" />
    </wsdl:message>
    <wsdl:message name="listdirRequest">
        <wsdl:part name="repository" type="xsd:string"></wsdl:part>
        <wsdl:part name="user" type="xsd:string"></wsdl:part>
        <wsdl:part name="password" type="xsd:string"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="listdirResponse">
        <wsdl:part name="listdirResponse" type="xsd:string"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="listjobsRequest">
        <wsdl:part name="repository" type="xsd:string"></wsdl:part>
        <wsdl:part name="user" type="xsd:string"></wsdl:part>
        <wsdl:part name="password" type="xsd:string"></wsdl:part>
        <wsdl:part name="directory" type="xsd:string"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="listjobsResponse">
        <wsdl:part name="listjobsResponse" type="xsd:string"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="executeJobRequest">
        <wsdl:part name="repository" type="xsd:string"></wsdl:part>
        <wsdl:part name="user" type="xsd:string"></wsdl:part>
        <wsdl:part name="password" type="xsd:string"></wsdl:part>
        <wsdl:part name="directory" type="xsd:string"></wsdl:part>
        <wsdl:part name="job" type="xsd:string"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="executeJobResponse">
        <wsdl:part name="result_code" type="xsd:int"></wsdl:part>
```

```
    <wsdl:part name="result_phrase" type="xsd:string"></wsdl:part>
    <wsdl:part name="time" type="xsd:string"></wsdl:part>
    <wsdl:part name="job_name" type="xsd:string"></wsdl:part>
    <wsdl:part name="creator" type="xsd:string"></wsdl:part>
    <wsdl:part name="creation_date" type="xsd:string"></wsdl:part>
    <wsdl:part name="description" type="xsd:string"></wsdl:part>
    <wsdl:part name="extended_description" type="xsd:string"></wsdl:part>
</wsdl:message>

<!-description des opérations du service-->
<wsdl:portType name="SOAP_Geokettle_list">
    <wsdl:operation name="listrep">
        <wsdl:input message="tns:listrepRequest" />
        <wsdl:output message="tns:listrepResponse" />
    </wsdl:operation>
    <wsdl:operation name="listdir">
        <wsdl:input message="tns:listdirRequest"></wsdl:input>
        <wsdl:output message="tns:listdirResponse"></wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="listjobs">
        <wsdl:input message="tns:listjobsRequest"></wsdl:input>
        <wsdl:output message="tns:listjobsResponse"></wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="executeJob">
        <wsdl:input message="tns:executeJobRequest"></wsdl:input>
        <wsdl:output message="tns:executeJobResponse"></wsdl:output>
    </wsdl:operation>
</wsdl:portType>

<!--liaison entre le portType, le format de message (RPC ou document_style) et les
     opérations-->
<wsdl:binding name="SOAP_Geokettle_listSOAP"
    type="tns:SOAP_Geokettle_list">
    <soap:binding style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http" />
    <wsdl:operation name="listrep">
        <soap:operation
            soapAction="http://scg.ulaval.ca/SOAP_Geokettle_list/listrep" />
        <wsdl:input>
            <soap:body
                namespace="http://scg.ulaval.ca/SOAP_Geokettle_list/"
                use="literal" />
        </wsdl:input>
        <wsdl:output>
            <soap:body
                namespace="http://scg.ulaval.ca/SOAP_Geokettle_list/"
                use="literal" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="listdir">
        <soap:operation
            soapAction="http://scg.ulaval.ca/SOAP_Geokettle_list/listdir" />
        <wsdl:input>
            <soap:body use="literal"
                namespace="http://scg.ulaval.ca/SOAP_Geokettle_list/" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"
                namespace="http://scg.ulaval.ca/SOAP_Geokettle_list/" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="listjobs">
        <soap:operation
            soapAction="http://scg.ulaval.ca/SOAP_Geokettle_list/listjobs" />
        <wsdl:input>
            <soap:body use="literal"
                namespace="http://scg.ulaval.ca/SOAP_Geokettle_list/" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"
                namespace="http://scg.ulaval.ca/SOAP_Geokettle_list/" />
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="executeJob">
        <soap:operation
            soapAction="http://scg.ulaval.ca/SOAP_Geokettle_list/executeJob" />
        <wsdl:input>
            <soap:body use="literal"
                namespace="http://scg.ulaval.ca/SOAP_Geokettle_list/" />
        </wsdl:input>
        <wsdl:output>
            <soap:body use="literal"
                namespace="http://scg.ulaval.ca/SOAP_Geokettle_list/" />
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>

<!--affectation d'une adresse à un binding-->
<wsdl:service name="SOAP_Geokettle_list">
    <wsdl:port binding="tns:SOAP_Geokettle_listSOAP"
        name="SOAP_Geokettle_listSOAP">
        <soap:address
```

```
                location="http://localhost:8080/axis/services/SOAP_Geokettle_listSOAP" />
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>
```

D.2 Messages SOAP pour l'opération listrep

D.2.1 Requête

La première partie du message est constituée des entêtes HTTP, la seconde est le message XML.

```
POST /axis/services/SOAP_Geokettle_listSOAP HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: 127.0.0.1:1234
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "http://scg.ulaval.ca/SOAP_Geokettle_list/listrep"
Content-Length: 371

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <listrep xmlns="http://scg.ulaval.ca/SOAP_Geokettle_list/">
            <listrepRequest xsi:nil="true" xmlns="" />
        </listrep>
    </soapenv:Body>
</soapenv:Envelope>
```

D.2.2 Réponse

De la même manière que pour la requête, la première partie du message est constituée des entêtes HTTP, la seconde est le message XML.

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Thu, 14 Feb 2008 14:21:10 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <listrepResponse
            xmlns="http://scg.ulaval.ca/SOAP_Geokettle_list/">
            <listrepResponse xmlns="">
                #1 : RepositoryKettle
            </listrepResponse>
        </listrepResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

D.3 Messages SOAP pour l'opération listdir

D.3.1 Requête

```
POST /axis/services/SOAP_Geokettle_listSOAP HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: 127.0.0.1:1234
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "http://scg.ulaval.ca/SOAP_Geokettle_list/listdir"
```

```
Content-Length: 448

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <listdir xmlns="http://scg.ulaval.ca/SOAP_Geokettle_list/">
            <repository xmlns="">RepositoryKettle</repository>
            <user xmlns="">charlotte</user>
            <password xmlns="">mdp</password>
        </listdir>
    </soapenv:Body>
</soapenv:Envelope>
```

D.3.2 Réponse

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Thu, 14 Feb 2008 14:21:13 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <listdirResponse
            xmlns="http://scg.ulaval.ca/SOAP_Geokettle_list/">
            <listdirResponse xmlns="">
                #1 : deprecated #2 : mysql_tutorial #3 :
                population_faits_division #4 : population_geom #5 :
                population_geom_complexe #6 : population_province
            </listdirResponse>
        </listdirResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

D.4 Messages SOAP pour l'opération listjobs

D.4.1 Requête

```
POST /axis/services/SOAP_Geokettle_listSOAP HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: 127.0.0.1:1234
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "http://scg.ulaval.ca/SOAP_Geokettle_list/listjobs"
Content-Length: 498

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <listjobs xmlns="http://scg.ulaval.ca/SOAP_Geokettle_list/">
            <repository xmlns="">RepositoryKettle</repository>
            <user xmlns="">charlotte</user>
            <password xmlns="">mdp</password>
            <directory xmlns="">/population_geom</directory>
        </listjobs>
    </soapenv:Body>
</soapenv:Envelope>
```

D.4.2 Réponse

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Thu, 14 Feb 2008 14:21:13 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
```

```
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <listjobsResponse
            xmlns="http://scg.ulaval.ca/SOAP_Geokettle_list/">
            <listjobsResponse xmlns="">
                #1 : JOB_Rebuild_aspatial #2 : JOB_Rebuild_cube #3 :
                JOB_Rebuild_spatial_dim #4 : JOB_update_spatial_dim #5 :
                JOB_update_spatial_dim_extractmaj
            </listjobsResponse>
        </listjobsResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

D.5 Messages SOAP pour l'opération execute

D.5.1 Requête

```
POST /axis/services/SOAP_Geokettle_listSOAP HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.4
Host: 127.0.0.1:1234
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: "http://scg.ulaval.ca/SOAP_Geokettle_list/executeJob"
Content-Length: 555

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <executeJob xmlns="http://scg.ulaval.ca/SOAP_Geokettle_list/">
            <repository xmlns="">RepositoryKettle</repository>
            <user xmlns="">charlotte</user>
            <password xmlns="">mdp</password>
            <directory xmlns="">/population_geom</directory>
            <job xmlns="">JOB_update_spatial_dim_extractmaj</job>
        </executeJob>
    </soapenv:Body>
</soapenv:Envelope>
```

D.5.2 Réponse

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml;charset=utf-8
Date: Thu, 14 Feb 2008 14:21:52 GMT
Connection: close

<?xml version="1.0" encoding="utf-8"?>
<soapenv:Envelope
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
        <executeJobResponse
            xmlns="http://scg.ulaval.ca/SOAP_Geokettle_list/">
            <result_code xsi:type="xsd:int" xmlns="">0</result_code>
            <result_phrase xmlns="">
                The job ran without problem
            </result_phrase>
            <time xmlns="">11</time>
            <job_name xmlns="">
                JOB_update_spatial_dim_extractmaj
                (JOB_update_spatial_dim_extractmaj (Thread-35))
            </job_name>
            <creator xmlns="">charlotte</creator>
            <creation_date xmlns="">
                2007/12/06 11:12:36.640
            </creation_date>
            <description xmlns="">update=delete+insert</description>
            <extended_description xmlns="">
                en extrayant uniquement les mises &#xE0; jour
            </extended_description>
        </executeJobResponse>
    </soapenv:Body>
</soapenv:Envelope>
```

Annexe E

Dimensions spatiales des cubes *population_provinces_spatial* et *population_provinces_spatial_detail* représentant le Canada

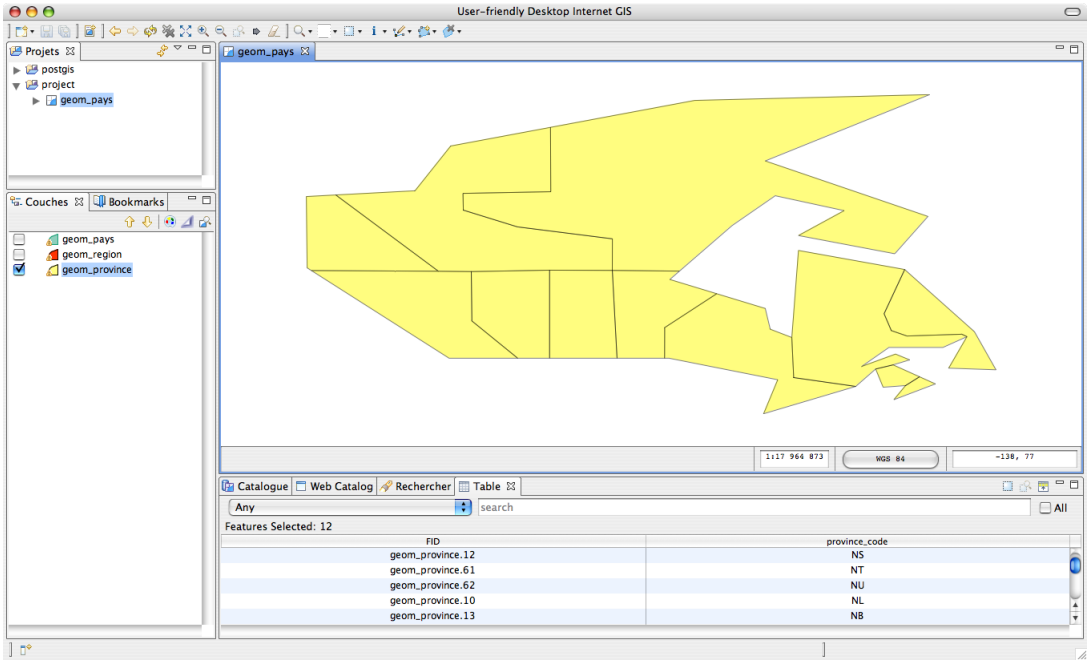


FIG. E.1 – Dimension spatiale à géométrie simplifiée (cube *population_provinces_spatial* (logiciel Udig)

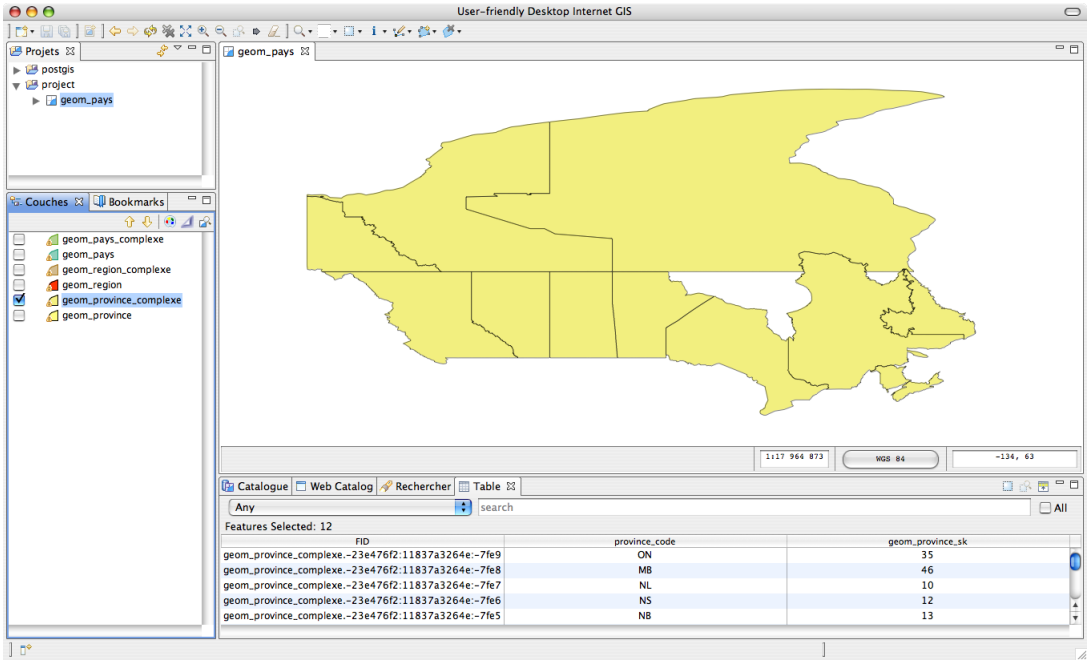


FIG. E.2 – Dimension spatiale à géométrie détaillée (cube *population_provinces_spatial_detail* (logiciel Udig)

Annexe F

Transformations et Jobs Kettle

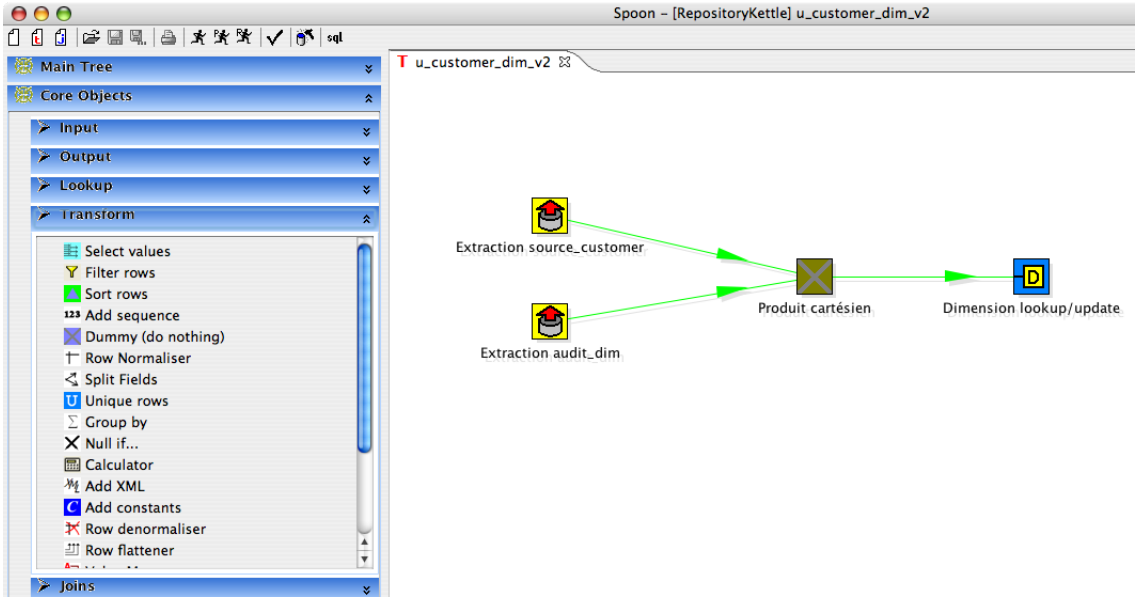


FIG. F.1 – Transformation pour les mises à jour récentes (naissance, modification) sur la dimension *customer* du cube *sales_orders*

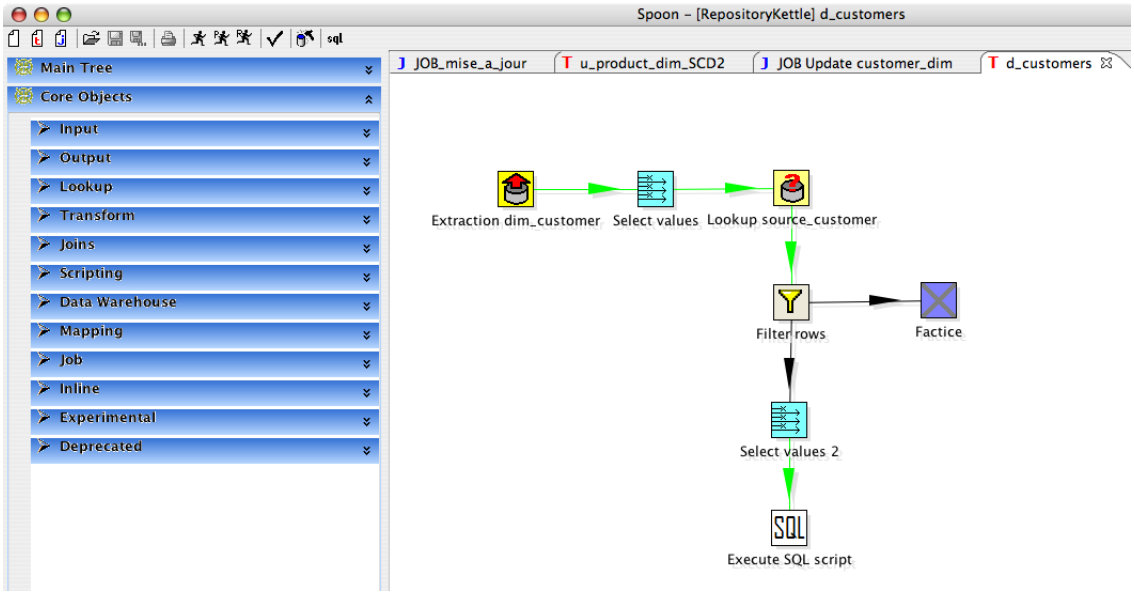


FIG. F.2 – Transformation pour les mises à jour récentes (mort) sur la dimension *customer* du cube *sales_orders*

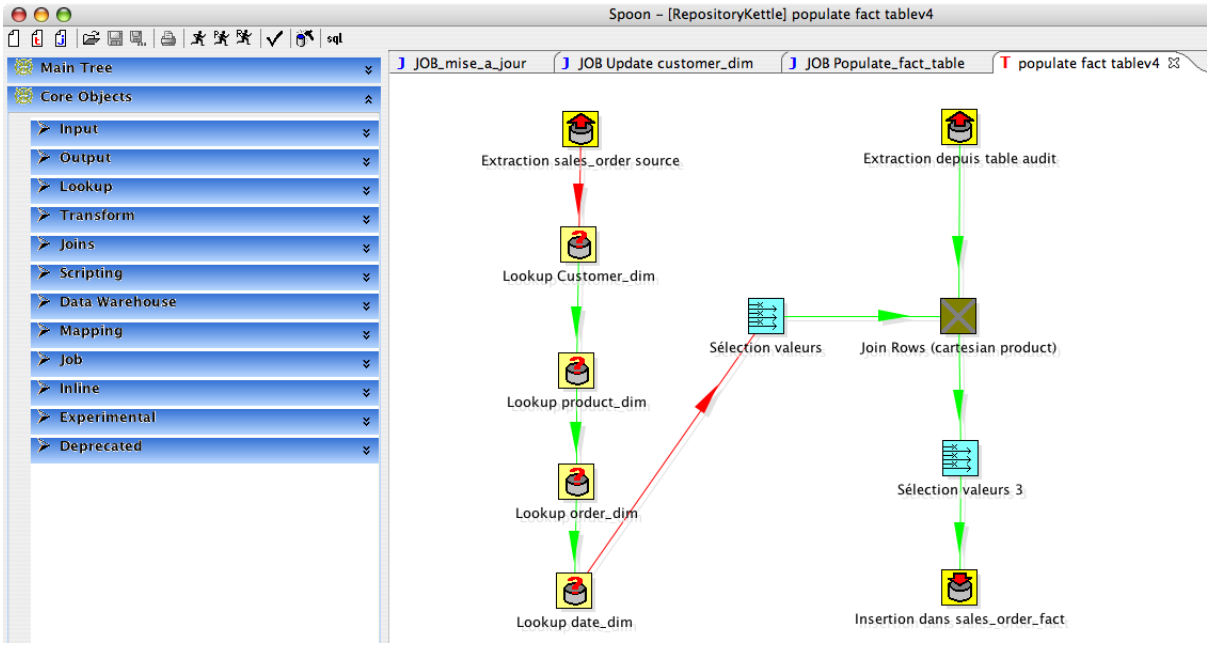


FIG. F.3 – Transformation pour les mises à jour récentes sur la table de faits *sales_order_fact* du cube *sales_orders*

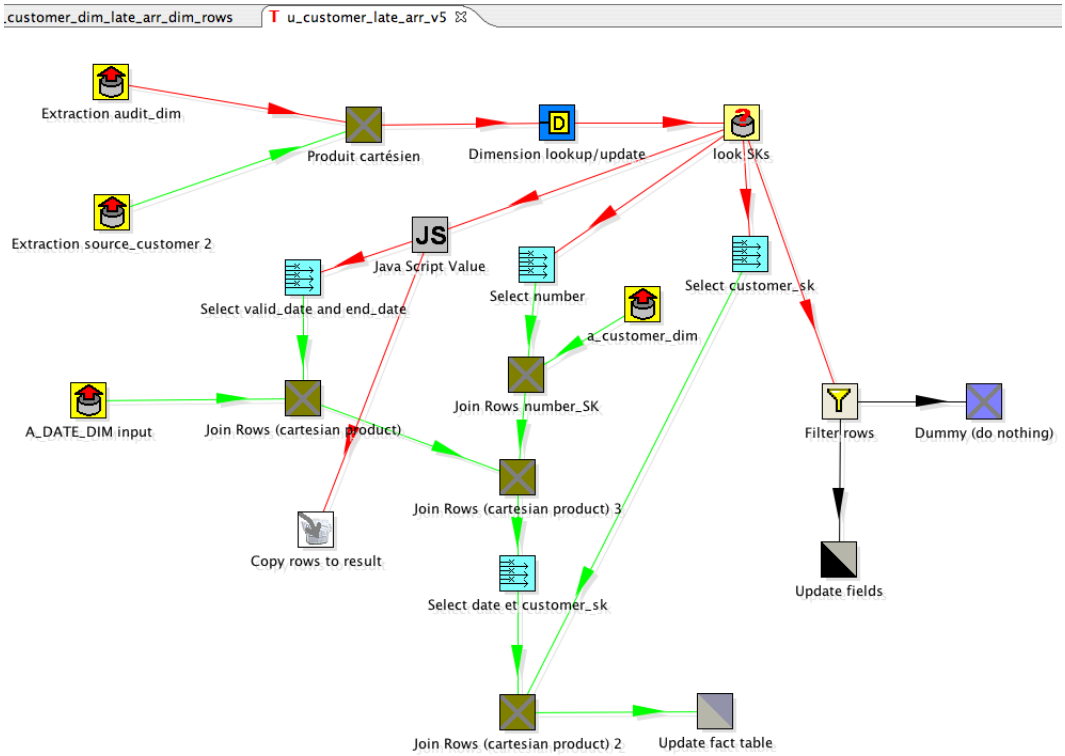


FIG. F.4 – Transformation pour les mises à jour tardives sur la dimension *customer* du cube *sales_orders*

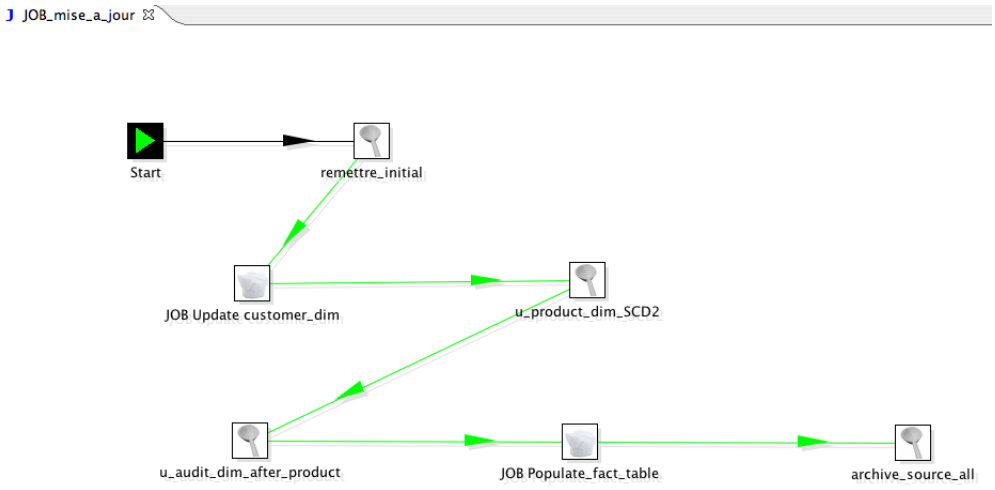


FIG. F.5 – Job pour la mise à jour du cube *sales_orders*

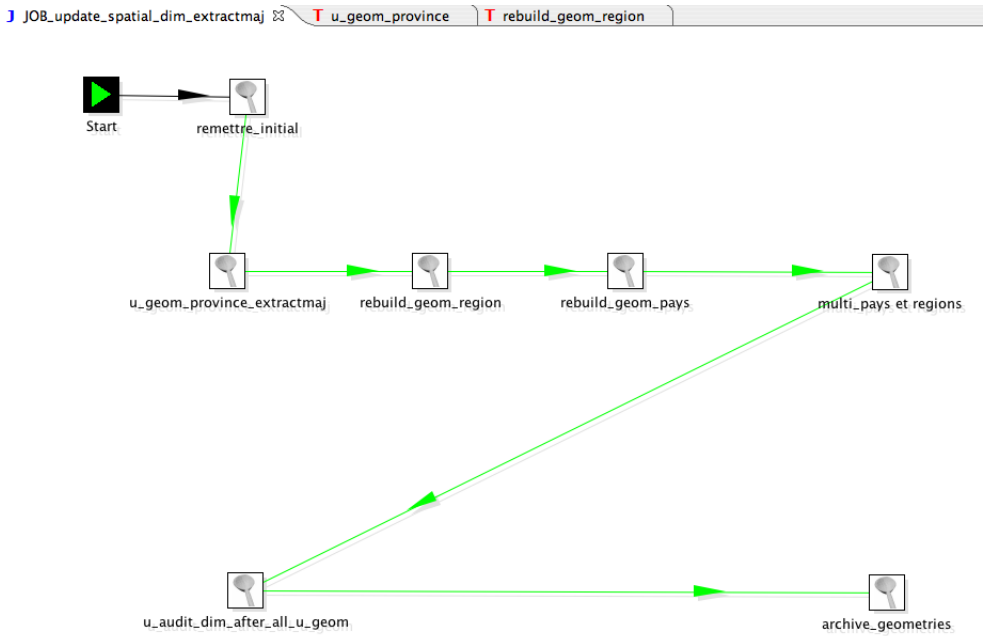


FIG. F.6 – Job pour la mise à jour de la dimension spatiale *province* des cubes *population_provinces_spatial* et *population_provinces_spatial_detail*

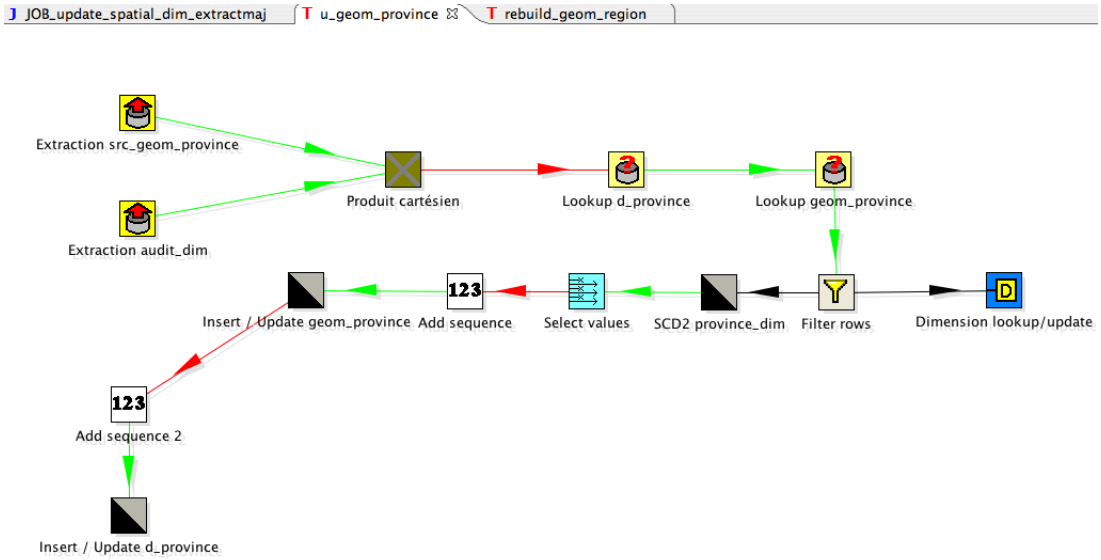


FIG. F.7 – Transformation pour la mise à jour du niveau *province* de la dimension spatiale *province* des cubes *population_provinces_spatial* et *population_provinces_spatial_detail*

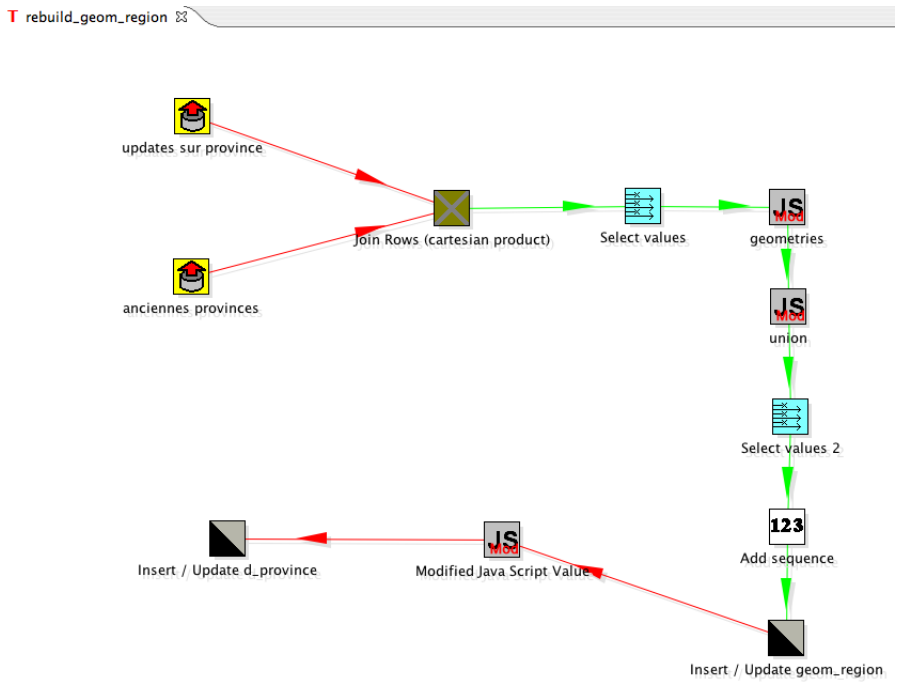


FIG. F.8 – Transformation pour la mise à jour du niveau *région* de la dimension spatiale *province* des cubes *population_provinces_spatial* et *population_provinces_spatial_detail*

Annexe G

Code PL/SQL pour le recalcul des versions

```
create or replace procedure u_customer_version(customer_number_param number) is
  -- update the versions numbers

     CURSOR cursor_SK IS
     SELECT customer_sk FROM a_customer_dim WHERE customer_number=
         customer_number_param order by effective_date;

     customer_dim_sk number;
     version_number number:=1;

  begin
   open cursor_sk;

     LOOP
       FETCH cursor_SK INTO customer_dim_sk;
       EXIT WHEN cursor_SK%NOTFOUND;
       UPDATE A_CUSTOMER_DIM
       SET version=version_number WHERE customer_sk=customer_dim_sk;
       version_number:=version_number+1;

     END LOOP;

  end u_customer_version;
```