Inférence de règles de contrôle d'accès pour assurer la confidentialité des données au niveau des vues matérialisées

Sarah Nait Bahloul



Numéro d'ordre : 242 - 2013 Année 2013

Université Claude Bernard Lyon 1
Laboratoire d'InfoRmatique en Image et Systèmes d'information
École Doctorale Informatique et Mathématiques de Lyon

Thèse de l'Université de Lyon

Présentée en vue d'obtenir le grade de Docteur,
spécialité Informatique

par

Sarah NAIT BAHLOUL

Inférence de règles de contrôle d'accès pour assurer la confidentialité des données au niveau des vues matérialisées

---

Thèse soutenue le 5 Décembre 2013 devant le jury composé de :

Mme.	Véronique Benzaken	Professeur à l'Université Paris Sud 11	Rapporteur
M.	Michaël Rusinowitch	Directeur de Recherche à l'INRIA, Université de Nancy	Rapporteur
Mme.	Régine Laleau	Professeur à l'Université Paris-Est Créteil	Examinatrice
M.	Farouk Toumani	Professeur à l'Université Blaise Pascal, Clermont Ferrand	Examinateur
M.	Mohand-Saïd Hacid	Professeur à l'Université Lyon 1	Directeur
M.	Emmanuel Coquery	Maître de Conférences à l'Université Lyon 1	Co-encadrant

Remerciements

En premier lieu, je tiens à adresser mes plus chaleureux remerciements à mon directeur de thèse M. Mohand-Saïd Hacid pour son aide et ses précieux conseils au cours de ces années. Sa compétence et sa rigueur scientifique m'ont beaucoup appris et je tiens tout particulièrement à exprimer mon admiration pour sa passion du travail. Je remercie également mon co-encadrant M. Emmanuel Coquery, pour son attention à tout instant sur mes travaux, pour toutes les réunions de travail qu'on a partagé, pour ses encouragements et ses conseils. J'ai pris un grand plaisir à travailler dans cet environnement et je leur adresse ma très profonde gratitude.

Je tiens à remercier les membres du jury de l'intérêt qu'ils ont porté à mes travaux. Je remercie Mme. Véronique Benzaken et M. Michaël Rusinowitch, d'avoir accepté d'être les rapporteurs de ce travail. Je remercie également Mme. Régine Laleau et M. Farouk Toumani d'avoir accepté de participer au jury.

Je remercie les membres de l'équipe Base de données, pour la dynamique au sein du groupe que ce soit sur le plan scientifique ou sur le plan humain. Je remercie également les membres du laboratoire LIRIS et tout particulièrement Mme Brigitte Gudayer pour sa joie de vivre qui a égayé nos journées.

Durant ces années de thèse, j'ai eu la chance de connaitre et de côtoyer de nombreuses personnes attachantes, qu'elles trouvent dans ces remerciements mon plaisir et ma joie d'avoir partagé d'aussi agréables moments. Je pense notamment à Lemya, Lotfi, Haïtang, Usman, Anis, Mustapha, Samy, Houssem, Hind, Sofia, Karim et Mehdi... J'en oublie certainement d'autres et je m'en excuse. J'exprime toute mon amitié à Hanane (pour toutes les aventures qu'on a vécu) et à Raafat (pour tout les débats enrichissants qu'on a eu).

Je remercie spécialement ma cousine Sonia, qui a su m'écouter et me réconforter à mon arrivée en France. Je lui dois beaucoup et lui souhaite toute la réussite dans ses études. Je remercie également mes beaux-parents, ainsi que Meriem et Karim (et leur adorable petite Dounia) pour leur soutien et de m'avoir accueilli dans leur famille à bras ouverts.

J'adresse toute mon affection à ma famille et en particulier à mes parents, ma soeur et mon frère, Naila et Sami. Malgré mon éloignement, leur amour et leur confiance me portent et me guident tous les jours. Je ne serai pas là où j'en suis sans eux.

Enfin, le dernier et pas des moindres, je voudrais remercier tout particulièrement mon cher époux Amine pour son amour, son soutien quotidien indéfectible et sa patience à mon égard. Merci d'être ce que tu es.

بقدرِ الكدِّ تكتسبُ المعالي ومن طلب العلا سهر الليالي
ومن رام العلا من غير كد أضاع العمر في طلب المحال

محمد بن إدريس الشافعي

La connaissance s'acquiert avec la persévérance, celui qui cherche la connaissance doit être prêt à veiller et celui qui la cherche sans dur labeur perdra sa vie à demander l'impossible.

Muhammad bin Idris ash-Shâfi'î

Résumé

Dans cette thèse, nous nous intéressons au problème de la confidentialité des données. Nous proposons une nouvelle approche pour faciliter l'administration des règles de contrôle d'accès pour assurer la confidentialité des données au niveau des vues matérialisées. Dans les bases de données relationnelles, une vue est une table virtuelle représentant le résultat d'une requête. À la différence d'une vue simple, une vue matérialisée stocke le résultat de la requête dans une table. Cette dernière peut être alors interrogée comme une table quelconque. Il est donc important d'y contrôler l'accès. Parmi les différents modèles proposés pour contrôler l'accès aux relations de base, nous nous basons dans notre approche sur l'utilisation des vues d'autorisations pour exprimer des règles de contrôle d'accès à grains fins. Nous proposons d'inférer, à partir des vues d'autorisations attachées aux tables de base, les vues d'autorisations qui doivent être attachées aux vues matérialisées.

Répondre à ce problème revient à répondre à un problème fondamental dans les bases de données relationnelles : Comment caractériser les informations calculables à partir de deux ensembles de vues ? Nous répondons à cette question en nous appuyant sur la réécriture de requêtes. Nous adaptons l'algorithme de réécriture de requêtes $MiniCon$ aux spécificités de notre problème et nous proposons l'algorithme $\mathcal{H}MiniCon^{+}$ qui se base sur un enchainement de réécritures. Nous nous intéressons aux vues représentées par des requêtes conjonctives en autorisant les égalités.

Nous nous sommes intéressés par la suite aux propriétés de cet algorithme. Nous démontrons que cet algorithme permet de calculer un ensemble de vues correctes, c.-à-d. toute information calculable à partir de l'ensemble de vues générées est calculable à partir de chacun des deux ensembles de vues de départ.

Afin d'étudier la terminaison de cet algorithme, nous définissons les arbres de réécritures générés par l'application de $\mathcal{H}MiniCon^{+}$ et étudions leur finitude. Nous caractérisons dans quel cas un arbre est fini et démontrons que l'approche est maximale, c.-à-d. toute information calculable à partir de chacun des deux ensembles de vues de départ est calculable à partir de l'ensemble de vues générées. Nous avons caractérisé dans quel cas l'algorithme risque de ne pas terminer c.-à-d. application infinie de l'algorithme de réécriture. Dans ce cas, il est impossible de déterminer la maximalité des résultats. Nous avons implémenté un prototype de l'approche et mené quelques expérimentations en utilisant des jeux de données synthétiques.

Abstract

In this thesis, we addresses the problem of data confidentiality. We propose a new approach to facilitate the administration of access control policies to ensure confidentiality of data in materialized views. In relational databases, a view is a virtual table representing the result of a query. Unlike a simple view, a materialized view persistently stores the data in a table. The latter can be queried like any other database table. We then need to control the access to the materialized view. Among the various models proposed for controlling access to base relations, we choose to express fine-grained access control through authorization views. We propose to infer, from the basic authorization views attached to the base tables, authorization views that will be attached to the materialized views.

Tackling this problem amounts to address a fundamental problem in relational databases : How to characterize computable information from two sets of views ? We handle this problem by resorting to query rewriting. We adapt the query rewriting algorithm *MiniCon* to the context of materialized views with access control and propose the $\mathcal{H}MiniCon^{+}$ algorithm which is based on successive rewritings. We mainly consider conjunctive queries with equalities.

We study the properties of our approach. We show that our algorithm can calculate a correct set of views, i.e. any computable information from the generated views is calculable from the two sets of views.

In order to prove the termination of our algorithm, we define rewriting trees generated by the application of $\mathcal{H}MiniCon^{+}$ and we study their features. We characterize in which case a tree is finite and show that the approach is maximal, i.e., any derivable information from the two sets of views can be derived from the set of generated views. We characterize in which case the algorithm could not terminate i.e., infinite application of the query rewriting algorithm. In this case, it is impossible to determine the maximality of results and this remains an open problem. We implemented a prototype of the approach and we led some experiments by using synthetic data sets.

Table des matières

Table des figures

Chapitre 1

Introduction

Sommaire

1.1 Contexte et Motivations

De par leur importance, les systèmes d'information sont devenus un point sensible des entreprises. Ajouté à cela, l'adoption de l'Internet comme infrastructure d'accès à distance aux ressources distribuées expose encore davantage les SGBDs aux attaques et les rend ainsi encore plus vulnérables. Un certain nombre de politiques sont mises en place pour pallier aux failles de sécurité.

Les failles de sécurité peuvent être catégorisées en trois types : (1) *l'accès non-autorisé* aux données qui résulte de la divulgation de l'information à des utilisateurs n'ayant pas le droit d'obtenir de telles informations, (2) *la modification incorrecte* des données qui peut être intentionnelle ou pas, mettant les bases de données dans un état incohérent et par conséquent toute utilisation de ces données devient compromettante, (3) *l'indisponibilité* des données aux utilisateurs légitimes ce qui rend les systèmes inefficaces au mieux, inutilisables au pire.

Afin de pallier à ces problèmes, une solution complète de sécurité doit satisfaire principalement trois besoins : 1) *Confidentialité* : permettre l'accès aux données qu'aux seuls utilisateurs légitimes, 2) *Intégrité* : empêcher la modification non autorisée ou incorrecte des données, et 3) *Disponibilité* : empêcher

et recouvrir les erreurs logicielles ou matérielles et les attaques malveillantes rendant les services indisponibles. D'autres besoins de sécurité peuvent être exprimés par les systèmes en fonction de leurs objectifs : L'*Intimité* qui relate du respect et de la protection de la vie privée des personnes, l'*authenticité* qui définit la propriété d'être vrai, la *non-répudiation* qui exprime l'impossibilité de nier d'être l'auteur d'une action, etc.

C'est dans le contexte de la gestion des données que se situe notre travail. Nous nous intéressons plus particulièrement à la confidentialité des données. Plusieurs recherches ont été effectuées et plusieurs techniques et modèles ont été proposés pour améliorer la sécurité des données et veiller sur leur confidentialité (voire, entre autres, [Mot89, RMSR04, OGM08, WYL+07]). La technique la plus répondue pour assurer la confidentialité des données est les mécanismes de contrôle d'accès. À chaque fois qu'un sujet essaye d'accéder à un objet, le mécanisme de contrôle d'accès vérifie les droits du sujet en ayant recours à un ensemble d'autorisations établi par l'organisation. Nous proposons dans notre travail une approche sûre et automatisée afin d'assurer la confidentialité des données via les mécanismes de contrôle d'accès dans les systèmes utilisant des vues matérialisées.

1.2 Problématique

Avec l'utilisation des grands systèmes de gestion de données comme les entrepôts [YKL97, TS99] ou les bases de données distribuées [CBHB09], de nouveaux défis de sécurité sont apparus [PP01, SG00]. Dans ce travail de thèse, nous nous sommes concentrés sur la sécurisation des vues matérialisées. Beaucoup d'organisations utilisent les SGBDs pour sauvegarder et gérer leurs informations et très souvent elles ont recours aux vues matérialisées. Dans les SGBDs relationnels, une *vue* est une relation virtuelle qui représente le résultat d'une requête. Une *vue matérialisée* enregistre les résultats retournés par la requête correspondante dans une table physique. Dans plusieurs contextes, les vues sont matérialisées dans le but d'optimiser les accès. Dans les entrepôts de données, elles sont utilisées pour calculer et sauvegarder des données agrégées. Dans un environnement distribué, les vues matérialisées sont utilisées pour répliquer les données dans des sites distribués et synchroniser les mises à

jour faites dans plusieurs sites. Par conséquent, les utilisateurs peuvent utiliser les vues matérialisées comme n'importe quelle autre table. Dans ce contexte, assurer la confidentialité des données au niveau des vues matérialisées est aussi important qu'assurer leur confidentialité au niveau des relations de base. La question qui se pose alors est la suivante : *Comment assurer la confidentialité des données au niveau des vues matérialisées ?* Dans Oracle [Win], le propriétaire de la vue matérialisée (celui qui crée la vue en ayant bien évidement les droits d'accès aux tables la définissant) établit les nouvelles règles de contrôle d'accès. Ainsi, les utilisateurs souhaitant accéder à la vue matérialisée ne sont soumis qu'aux nouvelles règles sans être soumis aux règles de contrôle d'accès des tables de base [1]. Cette approche n'assure pas la confidentialité et peut causer une divulgation d'informations. Pour cela, il suffit que le propriétaire définisse de nouvelles règles qui ne soient pas conformes aux règles de base. Nous souhaitons dans ce travail de thèse pallier à ce type de problèmes en automatisant la définition des règles de contrôle d'accès qui doivent être attachées aux vues matérialisées. Les nouvelles règles de contrôle d'accès aux vues matérialisées doivent être conformes aux règles de base, c.-à-d. l'administrateur (ou le propriétaire) doit définir de nouvelles règles de contrôle d'accès aux vues matérialisées en prenant en compte celles qui contrôlent l'accès aux tables de base [2]. Dans un système contenant une dizaine ou une centaine de tables contrôlées par une dizaine ou une centaine de règles, il devient impossible à un humain de faire face à un aussi grand ensemble de règles et de considérer celles qui sont les plus pertinentes. Il est nécessaire d'automatiser le processus de définition des règles de contrôle d'accès aux vues matérialisées. La figure 1.1 résume notre problématique : *Étant donnés un ensemble de relations de base (avec les règles de contrôle d'accès correspondantes) et un ensemble de définitions de vues matérialisées, comment définir un ensemble de règles de contrôle d'accès qui seront attachées aux vues matérialisées, de telle sorte que, requêter les vues matérialisées ne délivrera pas plus d'informations que requêter la base de données ?*

1. Tables qui sont impliquées dans la définition de la vue matérialisée.
2. Les tables qui sont utilisées pour définir les vues matérialisées.

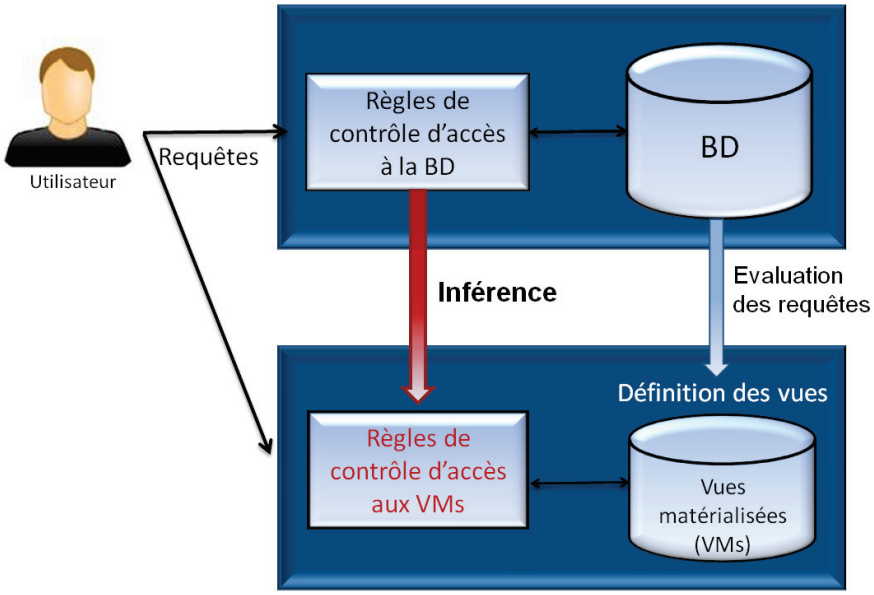

FIGURE 1.1 – Politiques de sécurité pour les vues matérialisées : Architecture du système.

1.3 Contributions

Nous proposons une approche sûre qui facilite l'administration des règles de contrôle d'accès pour assurer la confidentialité des données au niveau des vues matérialisées. Le but est d'automatiser le processus de définition des nouvelles règles d'accès aux vues matérialisées. Afin d'atteindre ce but, nous nous basons sur :

- **Les vues d'autorisations :** Plusieurs modèles ont été proposés pour représenter les règles de contrôle d'accès. Nous nous sommes tout particulièrement intéressés aux modèles basés sur les vues. Les règles de contrôle d'accès sont définies à travers un ensemble de vues, appelées, *vues d'autorisations*. À chaque (groupe d')utilisateur(s) correspond un ensemble de vues d'autorisations. Ces dernières définissent et retournent seulement les données accessibles à l'utilisateur. Ce modèle permet de définir des règles de contrôle d'accès à grains fins [3] et des règles de contrôle d'accès basées sur le contenu.
- **Requêtes conjonctives :** Les définitions des vues matérialisées ainsi que les vues d'autorisations sont considérées comme des requêtes sur

3. Possibilité de définir des accès uniquement à des cellules spécifiques d'une table.

les relations de base. Nous nous restreignons, dans notre cas, aux requêtes conjonctives avec égalités. Nous avons choisi d'exprimer les requêtes dans le langage Datalog [AHV95], connu pour sa flexibilité et sa faible complexité.

Ainsi, le but de l'approche est de définir un ensemble de vues d'autorisations qui garantit que les réponses d'une requête sur les vues matérialisées ne contiennent pas plus d'informations que les réponses de la même requête sur la base de données. Autrement dit, les informations retournées doivent être à la fois calculables à partir des vues matérialisées mais aussi autorisées par les vues d'autorisations de base. À partir de là, nous définissons notre problème indépendamment de la notion de sécurité, comme suit : Étant donnés une base de données et deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$ définies sur la base de données, comment calculer un troisième ensemble de vues $\mathcal{W}$ qui caractérise les informations calculables à partir des deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$? Répondre à cette question revient à calculer l'ensemble des vues d'autorisations ($\mathcal{W}$) qui garantiront la confidentialité des données au niveau des vues matérialisées ($\mathcal{V}_1$) conformément aux vues d'autorisations de base ($\mathcal{V}_2$). La figure 1.2 schématise notre problème fondamental.

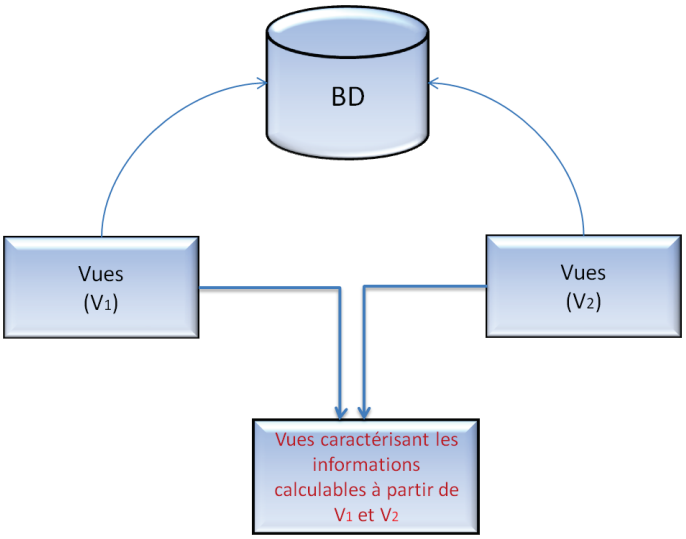

FIGURE 1.2 – Caractérisation des informations calculables à partir de deux ensembles de vues.

Afin de répondre à ce problème, nous présentons notre algorithme $\mathcal{H}MiniCon^{+}$, fondé sur le concept de réécriture de requêtes. Cette technique est utilisée dans

le domaine de la gestion de données pour répondre aux requêtes d'utilisateurs en utilisant un ensemble de vues. Nous définissons l'algorithme $\mathcal{H}MiniCon$ qui est l'adaptation de l'algorithme de réécriture de requêtes $MiniCon$ aux spécificités de notre problème. L'algorithme $\mathcal{H}MiniCon^+$ consiste en un enchainement d'application de l'algorithme $\mathcal{H}MiniCon$. L'algorithme $\mathcal{H}MiniCon^+$ prend en entrée deux ensembles de vues ($\mathcal{V}_1$ et $\mathcal{V}_2$) et retourne un ensemble de vues ($\mathcal{W}$) comme résultat. L'ensemble des vues $\mathcal{W}$ généré caractérise les informations calculables à partir des deux ensembles de vues donnés en entrée. Les vues générées $\mathcal{W}$ doivent satisfaire deux propriétés :

- **Correction :** L'ensemble des vues générées ne doit donner accès qu'aux seules informations accessibles à partir des deux ensembles de vues ($\mathcal{V}_1$ et $\mathcal{V}_2$). Autrement dit, si une information est retournée par $\mathcal{W}$ alors elle est calculable à la fois à partir de $\mathcal{V}_1$ et à partir de $\mathcal{V}_2$.
- **Maximalité :** L'ensemble des vues générées doit donner accès au maximum d'informations possible, de telle sorte que ces informations respectent la première propriété. C'est à dire, si une information est calculable à la fois à partir de $\mathcal{V}_1$ et de $\mathcal{V}_2$ alors elle doit être calculable à partir de $\mathcal{W}$.

Dans le but d'étudier les propriétés de notre algorithme, nous définissons *les arbres de réécritures* et *les arbres d'atomes*, deux structures qui découlent de l'enchainement successif de l'application de l'algorithme de réécriture adapté. À partir des propriétés de l'arbre, nous étudions la terminaison de notre algorithme. Nous caractérisons dans quel cas notre algorithme termine et est maximal et nous proposons une méthode pour détecter les cas où l'algorithme peut ne pas terminer [4].

Nous proposons une implémentation d'un prototype de l'approche. Nous réalisons nos expérimentations sur des jeux de données synthétiques. Nous proposons de traiter différentes configurations en entrée de l'algorithme en variant le nombre de vues, le nombre de relations de base utilisées, le nombre de jointures dans une vue... et étudions l'impact sur les performances du prototype.

4. Par conséquent, nous ne garantissons plus la maximalité des résultats.

1.4 Organisation du manuscrit

Le reste du manuscrit est organisé en deux parties.

La première partie présente notre approche. Plus précisément, le chapitre 2 introduit le concept de bases de données relationnelles : schéma, instance... Nous définissons le langage de requêtes que nous utiliserons tout au long de notre travail, à savoir les requêtes conjonctives avec égalités. La deuxième partie de ce chapitre traite des algorithmes de réécriture de requêtes. Nous détaillerons plus particulièrement les étapes de l'algorithme MiniCon qui constitue le socle de base de notre approche. Le chapitre 3 présente notre approche. Nous définissons l'algorithme $\mathcal{H}MiniCon$, une adaptation de l'algorithme MiniCon aux spécificités de notre problème. Nous présentons par la suite l'algorithme $\mathcal{H}MiniCon^{+}$ qui consiste en un enchainement d'applications de l'algorithme $\mathcal{H}MiniCon$ dans le but de générer l'ensemble des vues qui doivent être correctes et maximales. Nous définissons les arbres résultant de l'approche : arbres de réécritures et arbres d'atomes. Nous démontrons dans ce chapitre la correction de notre approche. Le chapitre 4 présente l'étude de terminaison de l'algorithme $\mathcal{H}MiniCon^{+}$. Nous nous intéressons aux caractéristiques des arbres et introduisons de nouveaux concepts : historique d'un noeud, noeud réel/virtuel... afin de démontrer la terminaison de l'approche sous certaines conditions. Nous caractérisons exactement les cas où l'algorithme est sûr de terminer. Dans le chapitre 5, nous étudions la maximalité de notre approche dans le cas où l'algorithme termine.

La deuxième partie de notre manuscrit traite du cadre applicatif de notre travail, à savoir, la confidentialité des données au niveau des vues matérialisées. Le chapitre 6 présente la sécurité dans les bases de données relationnelles. Nous décrivons les différents modèles théoriques et pratiques mis en place pour assurer la confidentialité des données. Nous discutons par la suite le modèle choisi dans le cadre de notre travail, à savoir, les vues d'autorisations. Nous exposons les différentes techniques d'utilisation des vues d'autorisations et les avantages et les inconvénients de chacune d'elles. Nous présentons les différentes solutions proposées pour assurer la confidentialité des données dérivées et leurs limites. Le chapitre 7 présente le prototype de notre approche ainsi

que les résultats expérimentaux réalisés sur des jeux de données synthétiques. Finalement, le chapitre 8 présente la conclusion et les perspectives de notre travail.

Première partie

Caractérisation des informations calculables à partir de deux ensembles de vues

Chapitre 2

Répondre aux requêtes en utilisant des vues

Sommaire

2.1 Requêtes

Nous présentons dans ce chapitre le langage logique Datalog et nous nous intéressons aux concepts autours des bases de données relationnelles [AHV95]. Nous nous intéressons plus particulièrement aux requêtes conjonctives avec égalités. Dans la deuxième section, nous présentons le problème de répondre aux requêtes en utilisant un ensemble de vues et nous nous intéresserons plus particulièrement aux algorithmes de réécriture de requêtes. Nous nous baserons dans notre approche sur l'algorithme MiniCon.

2.1.1 Datalog

L'intérêt principal de Datalog est sa flexibilité et sa simplicité. Il est utilisé à l'origine pour exprimer des requêtes déductives sur des bases de données relationnelles. Nous introduisons dans ce qui suit les notations utiles pour la définition d'une règle Datalog.

Nous supposons l'existence de trois types de symboles : les *variables* **Var** $= \{x, y, ...\}$, les *constantes* **Cons** $= \{a, b, ...\}$ et les *prédicats* **Pred** $= \{p, r, ...\}$. À chaque prédicat est associé un nombre d'arguments appelé *arité*. Un *terme* peut être soit une variable soit une constante. Un *tuple* est représenté par une séquence de n termes que l'on note $< x_1, ..., x_n >$ ou plus simplement $\bar{X}$. Si p est un prédicat d'arité n ($n \geq 1$) et $\bar{X}$ est un tuple d'arité n alors $p(\bar{X})$ est appelé *atome ordinaire*. Une règle Datalog peut contenir des atomes de comparaison (ex. $<$, $\leq$, $>$, $\geq$, $=$, $\neq$). Nous nous intéressons dans notre travail aux requêtes conjonctives en autorisant uniquement les égalités.

Définition 2.1 (Règle Datalog)**.** *Une règle Datalog est une expression de la forme :*

$$q(\bar{X}) \leftarrow p_1(\bar{X}_1), ..., p_n(\bar{X}_n).$$

où $q(\bar{X})$ et chaque $p_i(\bar{X}_i)$ pour $1 \leq i \leq n$ sont des atomes ordinaires. Chaque variable apparaissant dans $\bar{X}$ doit apparaitre au moins une fois dans $\bar{X}_1, ..., \bar{X}_n$. Un programme Datalog est un ensemble fini de règles Datalog.

2.1.2 Schéma relationnel et instance

Une distinction est faite entre le *schéma* d'une relation (resp. d'une base de données) qui spécifie la structure de la relation (resp. de la base de données) et l'*instance* d'une relation (resp. d'une base de données) qui spécifie le contenu de la relation (resp. de la base de données).

Un *schéma de relation* est un nom de relation associé à une arité. Soit r une relation d'arité n. Un *fait* sur r est une expression de la forme $r(a_1, ..., a_n)$ où $a_i \in$ **Cons** pour tout $i \in [1, n]$. Une *instance de relation* sur r, noté $\mathbf{I}_r$, est un ensemble fini de faits sur r. Un *schéma de base de données* $\mathbf{R}$ est un ensemble fini de schémas de relations. Une *instance de base de données* $\mathbf{I_R}$ est un ensemble fini représentant l'union des instances de relation sur r, pour chaque $r \in \mathbf{R}$.

2.1.3 Requêtes conjonctives

Nous considérons dans notre travail des requêtes conjonctives. Elles expriment les opérateurs de sélection, de projection et de jointure en autorisant uniquement les égalités.

Définition 2.2 (Requête conjonctive)**.** *Soit* ***R*** *un schéma de base de données. Une requête conjonctive est une règle Datalog de la forme :*

$$q(\bar{X}) \leftarrow r_1(\bar{X}_1), ..., r_n(\bar{X}_n).$$

où, chaque r_i *pour* $1 \leq i \leq n$ *est une relation de* ***R****. La tête* $q(\bar{X})$ *représente le résultat de la requête. Le corps de la requête contient un ensemble d'atomes ordinaires :* $r_1(\bar{X}_1), ..., r_n(\bar{X}_n)$*, aussi appelés sous-buts ordinaires, notés atomes*(q)*. Les atomes de jointure sont exprimés par plusieurs occurrences de la même variable ; c.-à-d. si une variable apparait à plusieurs endroits dans la requête, il faudrait que la valeur de la variable soit la même à chaque fois qu'elle apparait. L'ensemble des variables apparaissant dans* q *est noté vars*(q)*. Les variables apparaissant dans* $\bar{X}$ *sont appelées variables distinguées, ou variables de tête notées headvars*(q)*. Les variables qui apparaissent uniquement dans le corps sont appelées variables existentielles, notées Exist*(q)*.*

Intuitivement, une requête conjonctive q sur un schéma $\mathbf{R}$ est une fonction permettant à partir d'une instance $\mathbf{I}$ de construire un ensemble de faits définissant les réponses à la requête.

Définition 2.3 (Valuation)**.** *Étant donné un ensemble fini* V *de variables (*$V \subseteq$ ***Var****), une valuation* ψ *de* V *est une fonction totale de* V *vers* ***Cons****.*

Formellement, on peut définir l'interprétation des requêtes comme suit : soient q la requête donnée plus haut et $\mathbf{I}$ une instance de $\mathbf{R}$, l'instance de q est :

$$q(\mathbf{I}) = \{\psi(\bar{X}) |\ \psi \text{ est une valuation sur } vars(q),\ \psi(\bar{X}_i) \in \mathbf{I}(r_i), \text{ pour } 1 \leq i \leq n\}.$$

2.1.4 Vues, relations extentionnelles et intentionnelles

Une vue v est une requête nommée. Un ensemble de vues $\mathcal{V} = \{v_1, ..., v_n\}$ sur un schéma $\mathbf{R}$, définit un schéma de base de données sur lequel il est aussi

possible d'effectuer de nouvelles requêtes. Les vues peuvent être *matérialisées* (c.-à-d. une copie physique de la vue est sauvegardée et maintenue) ou bien rester *virtuelles* (c.-à-d. calculer la vue à la demande).

Considérons un schéma de base de données $\mathbf{R}$. Les relations $r_1, ..., r_n$ sont appelées *relations extentionnelles*, elles apparaissent uniquement dans le corps des règles. Supposons maintenant une vue v. Conceptuellement, cette requête définit une nouvelle relation avec un nouveau nom. Cette nouvelle relation peut être utilisée dans d'autres requêtes. Nous appelons ce type de relation *relations intentionnelles*. Il est a noté qu'il est tout à fait possible de définir une relation intentionnelle en termes d'autres relations intentionnelles. Illustrons cela par l'exemple suivant :

Exemple 1 Soit la vue (relation intentionnelle) suivante définie en termes des relations extentionnelles r_1 et r_2.

$$v_1(x, y) \leftarrow r_1(x, y), r_2(y, z).$$

Il est possible de définir des relations intentionnelles en termes de relations intentionnelles. v_2 est définie en utilisant v_1 :

$$v_2(x) \leftarrow v_1(x, y).$$

$\Diamond$

Remarque : Une requête q est dite exprimée uniquement sur $\mathcal{V}$, si toutes les relations apparaissant dans le corps de celle-ci appartiennent à $\mathcal{V}$ où $\mathcal{V}$ est un ensemble de relations intentionnelles (IDB).

2.1.5 Définitions et propriétés

Nous introduisons ici quelques définitions et propriétés que nous utiliserons tout au long de ce manuscrit.

Définition 2.4 (Inclusion de requêtes)**.** *Une requête q_1 est incluse dans une requête q_2, ou une requête q_2 subsume une requête q_1 noté $q_1 \sqsubseteq q_2$, si pour toute instance $\boldsymbol{I}$ de $\boldsymbol{R}$, les réponses de q_1 sont un sous ensemble des réponses de q_2, noté $q_1(\boldsymbol{I}) \subseteq q_2(\boldsymbol{I})$ [CM77].*

Définition 2.5 (Équivalence de requêtes). *Deux requêtes q_1 et q_2 sont équivalentes, et on note $q_1 \equiv q_2$ si $q_1 \sqsubseteq q_2$ et $q_2 \sqsubseteq q_1$.*

Le concept d'inclusion et d'équivalence de requêtes nous permet de comparer les résultats de deux requêtes posées sur la même instance de base de données. Une condition nécessaire et suffisante pour tester l'inclusion de requêtes est de trouver un morphisme entre les requêtes.

Définition 2.6 (Morphisme entre requêtes). *Un morphisme d'une requête q_1 vers une requête q_2 est une fonction h des termes de la requête q_1 vers les termes de la requête q_2 telle que l'image d'une constante c est la même constante c et l'image d'une variable est soit une variable soit une constante. L'application de ce morphisme induit une correspondance entre les atomes de q_1 et ceux de q_2, c.-à-d. la tête de q_1 devient la tête de q_2 et chaque atome de q_1 devient un certain atome de q_2. c.-à-d. $\forall g \in atomes(q_1),\ h(g) \in atomes(q_2)$.*

Définition 2.7 (Expansion). *Soit q une requête conjonctive de la forme :*

$$q(\bar{X}) \leftarrow v_1(\bar{X}_1), ..., v_n(\bar{X}_n).$$

définie en termes d'un ensemble de vues $\mathcal{V}$ où pour $1 \leq i \leq n$, $v_i(\bar{X}_i)$ est une requête conjonctive de la forme :

$$v_i(\bar{Y}_i) \leftarrow r_{1i}(\bar{Y}_{1i}), ..., r_{li}(\bar{Y}_{li}).$$

L'expansion de la requête q en utilisant les requêtes v_i, notée q^{exp}, est obtenue en remplaçant chaque $v_i(\bar{X}_i)$ par le corps de $\sigma_i(v_i)$ où σ_i est le mapping de $v_i(y_i^1...y_i^{n_i})$ vers $v_i(x_i^1, ..., x_i^{n_i})$ défini comme suit : $\sigma_i(y_i^k) = x_i^k$ si y_i^k est une variable de tête de v_i Sinon, $\sigma_i(y)$ est une variable fraiche.

Autrement dit, pour chaque atome $v_i(\bar{X}_i)$ de q, le mapping σ_i unifie les variables $\bar{Y}_i$ avec les variables $\bar{X}_i$ et unifie chaque autre variable dans $vars(v_i)$ avec des variables fraiches. Les variables fraiches sont le résultat du renommage des variables existentielles de v_i par des variables qui n'apparaissent nulle part ailleurs. L'expansion consiste donc à remplacer $v_i(\bar{X}_i)$ par le corps de $\sigma_i(v_i)$.

Exemple 2 Soit $\mathcal{V}$ l'ensemble des vues suivant :

$\mathcal{V} : v_1(x, z) \leftarrow r_1(x, y), r_2(y, z).$
$\quad v_2(z) \leftarrow r_3(y, z).$

Soit q la requête définie en termes de $\mathcal{V}$:

$q(w) \leftarrow v_1(w, t), v_2(t).$

L'expansion q^{exp} de la requête est définie par :

$q^{exp}(w) \leftarrow r_1(w, y_1), r_2(y_1, t), r_3(y_2, t).$ ◊

Nous notons dans la suite de ce manuscrit les variables fraiches introduites lors de l'expansion par un nom de variable suivi par un indice (ex. y_1, t_1).

2.1.6 Union de requêtes conjonctives

L'union de requêtes conjonctives est exprimée par un ensemble de requêtes conjonctives ayant la même tête [SY78]. Par exemple, l'union de requêtes conjonctives $\mathcal{Q}$ sur un schéma de base de données $\mathbf{R}$ est l'ensemble des requêtes conjonctives noté :

$$\mathcal{Q} = \{q_1(\bar{X}), ..., q_n(\bar{X})\}$$

L'interprétation de l'union de requêtes conjonctives est définie comme suit : Soient $\mathcal{Q}$ l'union de requêtes conjonctives et $\mathbf{I}$ l'instance de $\mathbf{R}$, l'instance de $\mathcal{Q}$ est définie par :

$$\mathcal{Q}(\mathbf{I}) = \bigcup \{q_i(\mathbf{I}) \mid q_i \in \mathcal{Q}\}$$

Définition 2.8 (Inclusion de requêtes)**.** *Soient $\mathcal{Q}_1$ et $\mathcal{Q}_2$ deux unions de requêtes conjonctives. On a $\mathcal{Q}_1 \sqsubseteq \mathcal{Q}_2$ si et seulement si pour tout $q_i \in \mathcal{Q}_1$, il existe une requête $q_j \in \mathcal{Q}_2$ telle que $q_i \sqsubseteq q_j$.*

2.2 Réécriture de requêtes

Le problème de répondre aux requêtes en utilisant des vues, aussi appelé réécriture de requêtes en utilisant des vues, a été principalement abordé dans les applications d'optimisation de requêtes [YL87, CKPS95], maintien de l'indépendance physique des données [TSI94], intégration des données [LRO96,

Ull97] et conception des entrepôts de données [GMR95]. D'une manière informelle, le problème est le suivant : étant donnés une requête q définie sur un schéma de base de données et un ensemble de définitions de vues $\mathcal{V}$ sur le même schéma, est-il possible de répondre à la requête q en utilisant seulement les réponses des vues $\mathcal{V}$? Ou bien, quel est l'ensemble de tuples maximal dans la réponse de q que nous pouvons obtenir des vues $\mathcal{V}$? Si nous pouvons accéder aux relations de la base de données et aux vues, quel est le meilleur plan d'exécution pour répondre à la requête q ? Nous présentons ici les définitions utilisées dans [Hal01]. Il existe deux principaux contextes où le problème de répondre aux requêtes en utilisant les vues a été considéré. Dans le premier contexte, où le but est l'optimisation des requêtes ou la maintenance de l'indépendance physique des données, c'est la notion d'équivalence de requêtes qui est importante.

La réécriture d'une requête q en utilisant un ensemble de vues $\mathcal{V}$ consiste à reformuler la requête, exprimée initialement en termes de relations extensionnelles (EDB), en une requête q' exprimée uniquement en termes de $\mathcal{V}$ (ensemble de relations intentionnelles(IDB)). La requête q' est alors appelée réécriture de la requête q. La réécriture q' représente les réponses à la requête q en utilisant uniquement les réponses de l'ensemble $\mathcal{V}$.

Définition 2.9. *Soient q une requête et $\mathcal{V} = \{v_1, ..., v_n\}$ un ensemble de vues définies sur le même schéma. Une réécriture q' est une réécriture équivalente à q si q' est exprimée uniquement sur $\mathcal{V}$ et $q' \equiv q$.*

Afin de déterminer si deux requêtes sont équivalentes, il faudrait comparer leur expansion (Définition 2.7). En effet, deux vues peuvent avoir des noms différents mais ont la même définition [1].

Dans le deuxième contexte, qui est l'intégration de données, les vues décrivent un ensemble de sources de données autonomes et hétérogènes qui peuvent devenir indisponibles. Dans ce cas, le système essaye de trouver les réécritures qui retournent le maximum de données possible à partir des vues. On parle ici de notion de réécriture maximalement-incluse.

Définition 2.10. *Soient q une requête et $\mathcal{V} = \{v_1, ..., v_n\}$ un ensemble de vues, définies sur le même schéma. Une réécriture q' est une réécriture maximalement-incluse dans une requête q par rapport à un langage de requêtes $\mathcal{L}$ si :*

1. Retournent le même résultat.

- $q' \in \mathcal{L}$ *est exprimée uniquement sur des vues de* $\mathcal{V}$
- $q' \sqsubseteq q$
- *il n'existe pas* $q'' \in \mathcal{L}$*, réécriture de* q *telle que :* $q' \sqsubset q'' \sqsubseteq q$.

À la différence des réécritures équivalentes, les réécritures maximalement-incluses dépendent de l'expressivité du langage de requêtes utilisé. Une réécriture peut être maximalement-incluse dans une requête par rapport à un langage $\mathcal{L}_1$ et n'être qu'incluse par rapport à un autre langage plus expressif $\mathcal{L}_2$.

La réécriture maximalement-incluse dans une requête conjonctive peut être le résultat d'une union de requêtes conjonctives. Prenons un exemple simple pour illustrer cet aspect.

Exemple 3 Soit le schéma d'une base de données hospitalière. Les relations $patients(x, y)$ et $docteurs(z, v, w)$ contiennent respectivement les informations sur des patients et sur des médecins d'un hôpital. La relation $traitements(x, z)$ contient l'identifiant des patients et l'identifiant des médecins traitants. Les relations $urgence(x)$ et $consult(x)$ contiennent les identifiants des patients admis dans le service urgence ou pour une simple consultation. La requête suivante récupère une partie des informations sur des patients :

$q(x) \leftarrow patients(x, y), docteurs(z, t, w), traitements(x, z).$

Supposons des vues définies sur le même schéma. La vue v_1 (resp. v_2) contient une partie des informations des patients du service urgence (resp. service consultation).

$v_1(x') \leftarrow patients(x', y'), docteurs(z', t', w'), traitements(x', z'), urgence(x').$
$v_2(x') \leftarrow patients(x', y'), docteurs(z', t', w'), traitements(x', z'), consult(x').$

Dans cet exemple, la réécriture maximalement-incluse de la requête q est l'union des deux réécritures suivantes :

$q'_1(x) \leftarrow v_1(x).$
$q'_2(x) \leftarrow v_2(x).$

Ces dernières présentent les différentes manières d'obtenir les résultats des différentes sources. Le résultat ne retournant que les informations sur les patients de seulement deux services, l'union de ces deux requêtes ne peut être

équivalente à la requête initiale. ◊

La recherche des réécritures maximalement-incluses dans le cas des requêtes conjonctives en autorisant uniquement les égalités est restreinte à une recherche dans un espace fini. L'idée est de considérer chaque conjonction possible de n vues, n étant le nombre d'atomes de la requête. Nous présentons dans ce qui suit les algorithmes proposés dans la littérature pour produire des réécritures de manière plus efficace qu'une recherche exhaustive. Nous décrivons les algorithmes dans le cas de requêtes et vues en autorisant uniquement les égalités.

2.2.1 L'algorithme Bucket

L'idée principale de l'algorithme Bucket est de réduire le nombre de réécritures qui doivent être étudiées en considérant chaque atome de la requête séparément afin de déterminer les vues contributives pour chaque atome [LRO96]. On appelle vue contributive une vue qui peut être utilisée pour répondre à l'atome. L'algorithme procède en deux étapes : (1) Construction d'un panier pour chaque atome de la requête contenant les vues contributives pour sa réécriture, et (2) Construction de toutes les requêtes conjonctives possibles en prenant une vue de chaque panier. Le résultat final est l'union de toutes les requêtes conjonctives qui sont incluses dans la requête initiale.

Dans sa première phase, l'algorithme essaie de trouver l'ensemble des vues qui sont contributives pour chaque atome de la requête. Une vue v est dite contributive pour un atome gd'une requête q si les conditions suivantes sont vérifiées :

- v contient un atome g' tel que g peut être unifié avec g'. L'unification est faite sur des atomes ayant le même nom de prédicat en tenant compte de la position des variables. Elle consiste a construire un mapping θ de $vars(g)$ vers $vars(g')$,
- Si x est une variable de tête dans la requête q et x apparait dans l'atome g, alors $\theta(x)$ doit apparaître comme variable de tête dans v.

Dans la phase de construction des paniers, si un atome g de la requête q s'unifie avec plusieurs atomes d'une vue v, alors le panier correspondant à g contiendra plusieurs occurrences de v.

La deuxième étape de l'algorithme consiste à calculer les réécritures conjonctives de la requête. Il s'agira dans cette étape de calculer le produit cartésien des paniers. Pour chaque combinaison, l'algorithme construit une réécriture conjonctive et vérifie si elle est contenue dans la requête. Si c'est le cas, l'algorithme retourne la réécriture comme résultat. Le résultat de l'algorithme est une union de réécritures conjonctives.

Exemple 4 Prenons la requête et l'ensemble des vues suivants pour illustrer le fonctionnement de l'algorithme.

$q(x) \leftarrow r_1(x,y), r_1(y,x), r_2(x,y).$
$v_1(x') \leftarrow r_1(x',y'), r_1(y',x').$
$v_2(x',y') \leftarrow r_2(x',y').$
$v_3(x',y') \leftarrow r_1(x',z'), r_1(z',y'), r_2(x',z').$

L'algorithme créera les paniers suivants :

$r_1(x,y)$	$r_1(y,x)$	$r_2(x,y)$
$v_1(x)$	$v_1(x)$	$v_2(x,y)$
$v_3(x,y)$	$v_3(x,y)$	$v_3(x,y)$

On note ici qu'il est possible d'unifier l'atome $r_1(x,y)$ de la requête avec l'atome $r_1(y',x')$ de la vue v_1, avec le mapping $\theta = \{x \rightarrow y',\ y \rightarrow x'\}$. Cependant, l'algorithme n'inclut pas la tête de la vue $v_1(y)$ dans les paniers parce qu'une des conditions de l'algorithme est que l'image de chaque variable de tête de la requête à travers θ doit être une variable de tête de la vue. Dans ce cas, x est une variable de tête dans q mais $\theta(x) = y'$ n'est pas une variable de tête dans v_1.

Dans la deuxième étape, l'algorithme calcule les produits cartésiens des paniers pour générer les requêtes conjonctives. En essayant de combiner la vue v_1 avec les autres vues, l'algorithme déduit qu'il ne peut pas réaliser cette combinaison. En effet, étant donné que la première étape consiste à prendre séparément chaque atome, l'algorithme oublie certaines interactions importantes entre les atomes. Si on considère la vue v_1 et le mapping de l'atome $r_1(x,y)$ à l'atome $r_1(x',y')$ de la vue, comme suit :

$Q(x) \leftarrow\ r_1(x,y),\ r_1(y,x), r_2(x,y).$
$\qquad\qquad\quad \downarrow \qquad \downarrow \qquad\quad ?$
$v_1(x') \leftarrow r_1(x',y'), r_1(y',x').$

Il est possible d'unifier y et y' et de satisfaire les deux atomes r_1 des deux clauses. Cependant, il est impossible de réaliser la jointure avec l'atome r_2 car la variable y n'est pas projetée par la vue. Mais l'algorithme ne détecte l'impossibilité d'utiliser la vue qu'à cette étape. De plus, un des inconvénients majeur de l'algorithme est son incapacité à détecter le fait qu'il doit utiliser la même vue pour couvrir plusieurs atomes de la requête. Dans cet exemple, il peut utiliser la vue v_1 pour couvrir les deux atomes r_1 de la requête. Ce qui le rendrait plus efficace en termes de performance. ◊

2.2.2 L'algorithme des règles inversées

Le fonctionnement de l'algorithme des règles inversées [Qia96, DG97] se base sur la construction d'un ensemble de règles qui inversent les définitions des vues, c.-à-d. définir les règles qui montrent comment calculer l'ensemble des tuples des relations de base à partir des tuples des vues. Le principe de construction des règles est le suivant :

- Pour chaque vue v_i définie comme suit : $v_i(\bar{X}) \leftarrow r_1(\bar{X}_1)...r_n(\bar{X}_n)$, construire n règles telles que la tête de la règle R_j pour $j = 1..n$ est l'atome $r_j(\bar{X}_j)$ du corps de la vue v_i et le corps de la règle R_j est la tête de la vue v_i.
- Remplacer par des fonctions Skolem, toutes les variables existentielles de la définition de v_i qui se retrouvent dans les têtes des règles. La même fonction Skolem est utilisée pour représenter toutes les occurrences de la même variable.
- L'ensemble des règles inversées est l'union des règles produites par chaque vue.

Exemple 5 Continuons avec l'exemple précédent. Les règles produites de l'ensemble des vue $\{v_1, v_2, v_3\}$ sont les suivantes :

$R_1 : r_1(a, f_1(x')) \leftarrow v_1(x').$
$R_2 : r_1(f_1(x'), x') \leftarrow v_1(x').$
$R_3 : r_2(x', y') \leftarrow v_2(x', y').$
$R_4 : r_1(x', f_2(x', y')) \leftarrow v_3(x', y').$
$R_5 : r_1(f_2(x', y'), y') \leftarrow v_3(x', y').$

$R_6 : r_2(x', f_2(x', y')) \leftarrow v_3(x', y').$ ◊

Une fois les règles inversées obtenues, la requête initiale est réécrite. La réécriture d'une requête en utilisant l'ensemble des règles inversées consiste à évaluer cette requête sur les tuples produits par ces règles. Il est à noter que l'avantage principal de l'algorithme des règles inversées est le fait que les règles sont calculées à l'avance en un temps polynomial et ce calcul est indépendant de la requête à réécrire. Néanmoins, cette approche présente quelques limites quant à son application. Le premier inconvénient est de devoir répéter certains calculs qui étaient déjà réalisés par la vue. Par exemple : la vue v_3 réalise déjà la jointure entre les trois relations, chose qui devrait être recalculée pour évaluer la requête en utilisant les règles inversées. Le deuxième inconvénient est de devoir utiliser les différentes combinaisons de toutes les règles même si certaines règles ne sont pas contributives à l'évaluation de la requête. Cette étape est l'équivalent du produit cartésien réalisé par l'algorithme de bucket.

2.2.3 L'algorithme MiniCon

Comme l'algorithme Bucket, l'algorithme MiniCon [PL00] fonctionne en deux étapes qui sont : la recherche des vues contributives pour la réécriture de la requête et la combinaison de ces vues afin d'obtenir des réécritures de requête. Dans la première phase, l'algorithme détermine un mapping partiel entre un atome g de la requête q et un atome g' d'une vue v. À la différence de l'algorithme Bucket qui construit des paniers indépendants pour chaque atome, l'algorithme MiniCon examine les variables de jointure de la requête afin de déterminer l'ensemble minimal des atomes de v qui doivent être unifiés avec des atomes de q étant donné que g est unifié avec g'. Autrement dit, l'algorithme vérifie que toutes les jointures réalisées au niveau de la requête q sont soit vérifiées par la définition de la vue, soit exprimées sur des variables distinguées de la vue afin d'être satisfaites ultérieurement. Pour chaque vue, l'algorithme note les atomes de la requête qui peuvent être couverts par cette vue. Les informations décrivant les mappings entre une vue v et une requête q et les atomes de la requête couverts par cette vue sont appelées "MiniCon Description" (MCD). Une fois les MCDs construits, l'algorithme les combine afin d'obtenir des réécritures de la requête q. L'algorithme construit les MCDs

de telle manière qu'il n'ait plus besoin d'explorer certaines combinaisons et ainsi produire directement des réécritures contenues dans la requête. En effet, la vérification de l'inclusion des réécritures dans la requête est faite en amont, lors de la construction des MCDs. Ainsi, l'algorithme gagne en performance comparativement à l'algorithme Bucket.

Notre approche se basant sur l'algorithme MiniCon, nous détaillons dans ce qui suit les deux étapes de l'algorithme : La formation des MCDs et leur combinaison.

2.2.3.1 Formation des MCDs :

L'idée principale de cette étape est de construire au fur et a mesure des fragments du morphisme de la requête vers sa réécriture. La manière dont sont construits ces fragments permettra de les combiner naturellement.

L'algorithme définit pour chaque atome de la requête les vues qui peuvent le couvrir. Un sous but g' d'une vue v couvre un atome g de la requête q s'il existe un mapping δ de $vars(q)$ vers $vars(v)$ tel que : $\delta(g) = g'$. Plus précisément, l'algorithme considère les mappings de la requête vers des spécialisations des vues, où les variables de tête peuvent être unifiées. À chaque MCD est alors associé un homomorphisme de tête. Un homomorphisme de tête h sur une vue v est un mapping de $vars(v)$ vers $vars(v)$ qui est l'identité sur les variables existentielles et qui peut unifier des variables distinguées [2]. Un MCD C d'une requête par rapport à une vue v est représenté par un quadruplet $(h_C, v(\bar{Y})_C, \varphi_C, G_C)$ où :

- h_C est l'homomorphisme sur les variables de tête de v.
- $v(\bar{Y})_C$ est le résultat de l'application de h_C sur la tête de v.
- φ_C est le mapping (unification) partiel des variables de la requête vers les variables de la vue.
- G_C est l'ensemble des atomes de q qui sont couverts par la vue v en utilisant le mapping φ_C.

Pour utiliser un MCD C dans la génération d'une réécriture, les conditions suivantes doivent être vérifiées :

C_1. Pour chaque variable de tête x de la requête q qui est dans le domaine de φ_C, $\varphi_C(x)$ est une variable de tête de $h_C(v)$.

2. Les variables qui apparaissent dans la tête de la vue.

C_2. Si $\varphi_C(x)$ est une variable existentielle dans $h_C(v)$, alors pour chaque atome g de q qui inclut x, (1) toutes les variables de g sont dans le domaine de φ_C, et (2) $\varphi_C(g) \in h_C(v)$.

C'est en construisant des MCDs qui vérifient la condition C_2 que l'algorithme MiniCon gagne en performance. En effet, cette condition garantit que si une variable de jointure n'est pas dans la tête, alors la jointure est réalisée au niveau de la vue du MCD. Cela permettra d'éliminer à cette étape toutes les vues qui ne peuvent pas être combinées par la suite et par conséquent le nombre de combinaisons possibles.

Exemple 6 Continuons avec l'exemple précédent pour illustrer la formation des MCDs. L'algorithme MiniCon détectera à l'étape de construction des MCDs que la vue v_1 ne peut pas être contributive, du fait que la variable y' n'est pas dans la tête de la vue et celle-ci ne couvre pas l'atome r_2. Les MCDs formés par l'algorithme sont les suivants :

$v(\bar{Y})$	h	φ	G
$v_2(x', y')$	$x' \to x', y' \to y'$	$x \to x', y \to y'$	3
$v_3(x', x')$	$x' \to x', y' \to x'$	$x \to x', y \to x'$	1,2,3

$\Diamond$

Il est important de dire que l'algorithme, lors de la formation des MCDs, inclura l'ensemble minimal des atomes couverts par la vue dans le but de satisfaire la condition C_2. Ceci afin de se focaliser dans la deuxième étape sur la construction de réécritures à partir d'un ensemble de MCDs qui couvrent un ensemble d'atomes exclusivement distincts.

2.2.3.2 Combinaison des MCDs :

L'idée principale de cette étape est de construire des réécritures conjonctives à partir des différentes combinaisons valides de MCDs. La réécriture finale est l'union des réécritures conjonctives.

Une combinaison valide de MCDs doit couvrir l'ensemble des atomes de la requête et que pour toute paire de MCDs, ils doivent couvrir des atomes disjoints. Autrement dit, Soient $C_1, C_2, ..., C_k$ des MCDs formant une réécriture candidate de q, les deux conditions suivantes doivent être vérifiées :

- $G_{C_1} \cup ... \cup G_{C_k} = atomes(q)$
- $\forall i, j, i \neq j, G_{C_i} \cap G_{C_j} = \emptyset$

Exemple 7 En reprenant l'exemple précédent, l'algorithme ne générera qu'une seule réécriture :

$$rw(x) \leftarrow v_3(x, x).$$

En effet, l'algorithme considérera la vue v_2 pour couvrir l'atome r_2, mais aucune autre vue ne peut couvrir les autres atomes de la requête. Par conséquent, la vue v_2 est écartée et seule v_3 est prise en compte puisqu'elle couvre les trois atomes. ◊

Nous présentons le théorème définit dans [PL00], qui garantit que l'algorithme MiniCon génère des réécritures maximallement-incluses dans le cas de requêtes et vues conjonctives.

Théorème 2.11. *Soient q une requête conjonctive et $\mathcal{V}$ un ensemble de vues conjonctives sans atomes de comparaisons ni constantes, l'algorithme MiniCon produit une union de requêtes conjonctive qui est une réécriture maximallement incluse dans la requête q en utilisant $\mathcal{V}$.*

Chapitre 3

Enchainement de réécritures

Sommaire

3.1 Introduction

Nous exposons dans ce chapitre notre approche pour répondre à notre problématique : *Comment déterminer, à partir de deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$ définies sur le même schéma, un ensemble de vues contenant toutes les informations que l'on peut récupérer à la fois à partir de $\mathcal{V}_1$ et à partir de $\mathcal{V}_2$?*. Plus formellement, nous décrivons notre problématique comme suit : Soient $\mathcal{R}$ un ensemble de relations extensionnelles, $\mathcal{V}_1$ et $\mathcal{V}_2$ deux ensembles de relations intentionnelles définies en termes des relations $\mathcal{R}$. L'objectif consiste à dériver un ensemble de relations intentionnelles $\mathcal{W}$ définies en termes des relations d'un des ensembles ($\mathcal{V}_1$ ou $\mathcal{V}_2$) tel que cet ensemble ne retourne pas plus d'informations que $\mathcal{V}_1$ et $\mathcal{V}_2$. De plus, il doit retourner le maximum d'informations possible. Une remarque importante est à faire sur l'objectif de notre approche. Nous ne souhaitons pas déterminer les tuples présents dans les deux ensembles de vues mais l'objectif est de déterminer quels sont les ensembles de tuples accessibles à la fois à partir de $\mathcal{V}_1$ et à partir de $\mathcal{V}_2$. Prenons l'exemple suivant pour illustrer cette différence.

Exemple 8 Soient les deux ensembles de vues définies en termes des relations de notre base de données hospitalière :

$\mathcal{V}_1 : v_{11}(y) \leftarrow patients(x, y), urgence(x).$
$\mathcal{V}_2 : v_{21}(y) \leftarrow patients(x, y).$

La vue v_{11} retourne les informations des patients admis dans le service des urgences et la vue v_{21} retourne les informations de tous les patients de l'hôpital. Supposons la vue suivante :

$\mathcal{W} : v(y) \leftarrow patients(x, y), urgence(x).$

Nous pouvons dire dans cet exemple que quel que soit l'instance de la base de données nous avons : $v \sqsubseteq v_{11}$ et $v \sqsubseteq v_{21}$. Nous remarquons par ailleurs qu'il est possible de déduire une information supplémentaire à partir de la vue v par rapport à la vue v_{21}. En effet, à partir de la vue v nous obtenons les tuples qui concernent seulement les patients du service des urgences. Même si ces tuples sont un sous ensemble des tuples retournés par la vue v_{21}, il n'y a aucun moyen de les calculer directement à partir de v_{21}. L'attribut x n'étant pas dans la tête de la vue v_{21}, il est impossible de filtrer les patients par rapport au service où ils ont été traités. De ce fait, en ayant accès à la vue v, l'utilisateur aura une information supplémentaire par rapport à ce à quoi il avait accès à partir de la vue v_{21}. C'est ce type de divulgation d'informations que nous souhaitons aussi prendre en considération.

Au final, dans cet exemple, il n'y a aucun ensemble de vues $\mathcal{W}$ qui peut définir l'ensemble de tuples qui peuvent être obtenus à la fois à partir de $\mathcal{V}_1$ et à partir de $\mathcal{V}_2$. ◊

Afin de caractériser la correction de l'approche, nous avons défini deux conditions nécessaires que l'ensemble de vues générées doit satisfaire : la correction et la maximalité des vues.

Définition 3.1 (Correction des vues)**.** *Soient $\mathcal{V}_1$ et $\mathcal{V}_2$ deux ensembles de vues définies sur le même schéma et $\mathcal{W}$ l'ensemble des vues calculées. $\mathcal{W}$ ne doit pas retourner des informations qui ne peuvent pas être retournées par l'un des ensembles. Plus formellement, pour chaque relation intentionnelle $q^{\mathcal{W}}$ définie en termes de $\mathcal{W}$, il existe une relation intentionnelle $q^{\mathcal{V}_1}$ définie en termes de*

$\mathcal{V}_1$ et une relation intentionnelle $q^{\mathcal{V}_2}$ définie en termes de $\mathcal{V}_2$ telles que :

$$q^{\mathcal{W}} \equiv q^{\mathcal{V}_1} \text{ et } q^{\mathcal{W}} \equiv q^{\mathcal{V}_2}$$

Définition 3.2 (Maximalité des vues)**.** *Les vues calculées doivent retourner le maximum d'informations possible, tout en étant correctes. Plus formellement, s'il existe une relation intentionnelle $q^{\mathcal{V}_1}$ définie en termes de $\mathcal{V}_1$ et une relation intentionnelle $q^{\mathcal{V}_2}$ définie en termes de $\mathcal{V}_2$, telles que :*

$$q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2}$$

alors il existe une relation intentionnelle $q^{\mathcal{W}}$ définie en termes de $\mathcal{W}$ telle que :'

$$q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2} \equiv q^{\mathcal{W}}$$

3.2 Double réécriture

Dans le but de répondre à notre problématique, nous nous basons sur l'algorithme de réécriture de requêtes MiniCon. L'idée générale de notre approche est d'utiliser l'algorithme à travers une série de doubles réécritures en utilisant les deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$ l'un après l'autre jusqu'à l'obtention de l'ensemble de vues $\mathcal{W}$ qui doit être correcte. Plus précisément, l'intérêt de la double réécriture en utilisant l'un après l'autre les deux ensembles $\mathcal{V}_1$ et $\mathcal{V}_2$ consiste à filtrer les ensembles de tuples qui sont accessibles à partir de $\mathcal{V}_1$ et à partir de $\mathcal{V}_2$.

La première étape réside dans la définition du point de départ : quels sont les ensembles de tuples de départ à partir desquels l'algorithme doit filtrer ? Le choix ici se porte arbitrairement sur l'un des deux ensembles de vues $\mathcal{V}_1$ ou $\mathcal{V}_2$. C'est donc sans perdre de généralité que nous choisissons de prendre comme point de départ l'ensemble $\mathcal{V}_1$ pour le déroulement de notre algorithme. Nous définissons donc un ensemble de requêtes $\mathcal{Q}$ de la manière suivante : pour chaque vue v_{1i} de $\mathcal{V}_1$ on définit une requête q sur v_{1i} comme suit :

- $headvars(q) = headvars(v_{1i})$
- $atomes(q) = \{v_{1i}\}$

Dans l'exemple 8 (page 28), $\mathcal{Q}$ est définie comme suit :
$\mathcal{Q} : q(y) \leftarrow v_{11}(y)$.

Après la définition de l'ensemble de départ $\mathcal{Q}$, l'algorithme se base sur l'algorithme de réécriture de requêtes pour réécrire chaque q dans $\mathcal{Q}$ en utilisant l'ensemble de vues $\mathcal{V}_2$. L'algorithme retourne $\mathcal{RW}$, l'ensemble des réécritures qui représentent les ensembles de tuples de q accessibles à partir de $\mathcal{V}_2$.

Par définition, les tuples appartenant à $\mathcal{RW}$ appartiennent à $\mathcal{V}_1$ (c'est notre ensemble de départ). Cependant, notre but est de calculer les ensembles de tuples qui sont accessibles à partir des deux ensembles de vues. En effet, comme il a été indiqué dans l'exemple 8, les deux ensembles de vues ne sont pas forcément définis sur les mêmes relations. La vue v_1 est définie sur les relations *patients* et *urgence* alors que la vue v_2 n'est définie que sur la relation *patients*. Il est donc possible d'utiliser certaines relations pour filtrer les tuples et ces relations, n'apparaissant pas dans le deuxième ensemble de vues, rendraient ainsi impossible de filtrer de la même manière et par conséquent ne pas pouvoir recalculer le même ensemble de tuples. Il est donc important de vérifier que les ensembles de tuples appartenant à $\mathcal{RW}$ peuvent être accessibles à partir de $\mathcal{V}_1$. Pour cela, l'algorithme réapplique la réécriture en prenant l'ensemble des requêtes $\mathcal{RW}$ comme des requêtes à réécrire en utilisant l'ensemble des vues $\mathcal{V}_1$. Nous notons $\mathcal{RW}'$ l'ensemble des réécritures générées par cette deuxième étape de réécriture.

De la même manière, la deuxième étape de réécriture peut introduire des relations sur lesquelles aucune vue n'est définie dans $\mathcal{V}_2$. Par conséquent, l'algorithme doit aussi vérifier que les ensembles de tuples de $\mathcal{RW}'$ sont aussi accessibles à partir de $\mathcal{V}_2$. Pour cela, l'algorithme réalise cette double réécriture jusqu'à atteindre un point où les ensembles de tuples des réécritures générées sont accessibles à la fois à partir de $\mathcal{V}_1$ et à partir de $\mathcal{V}_2$, c.-à-d. plus aucun filtrage n'est réalisé par l'algorithme de réécriture.

Afin d'atteindre notre objectif, nous nous basons sur une modification de l'algorithme MiniCon. Nous verrons dans la section suivante nos motivations pour les modifications apportées ainsi que l'impact sur le résultat d'application de l'algorithme.

3.3 $\mathcal{H}MiniCon$: Adaptation de l'algorithme MiniCon

Nous rappelons ici que le but d'un algorithme de réécriture de requêtes est de répondre à une requête en utilisant un ensemble de vues. Autrement dit, quels sont les tuples présents au niveau des vues et qui sont des réponses à la requête de départ ? Dans notre approche, nous souhaitons également déterminer les tuples présents au niveau des vues qui répondent à la requête et ceci même si c'est une réponse partielle [1] à la requête.

Nous présentons dans cette section notre algorithme de réécriture de requête $\mathcal{H}MiniCon$ qui découle de l'algorithme MiniCon en apportant deux modifications nécessaires pour répondre à notre problématique. Celles-ci concernent les variables de tête. La première consiste en un assouplissement de la condition sur les variables de tête lors de la formation des MCDs. La deuxième modification consiste à ajouter certaines variables à la tête des réécritures afin d'assurer la maximalité des vues. Ainsi, l'algorithme $\mathcal{H}MiniCon$ fonctionne en trois parties :

1. **Formation des MCDS :** Cette étape est identique à la formation des MCDs de l'algorithme MiniCon. L'algorithme construit des MCDs à partir des vues contributives. De la même manière, chaque MCD est représenté par un quadruplet (voir section 2.2.3.1, page 23). Cependant, dans notre algorithme, un MCD C peut être utilisé s'il vérifie seulement la condition C_2 définie dans l'algorithme MiniCon.

   La première modification consiste donc en l'assouplissement de la condition C_1. Autrement dit, si x est une variable de tête dans la requête, il n'est plus nécessaire de vérifier que $\varphi_C(x)$ est une variable de tête de $h_C(v)$.

2. **Combinaison des MCDs :** Aucune modification n'est apportée à cette étape. De la même manière qu'avec l'algorithme MiniCon, une combinaison valide de MCDs doit couvrir l'ensemble des atomes de la requête et que les MCDs de toute paire [2] doivent couvrir des atomes disjoints.

3. **Détermination des variables de tête :** Après la combinaison des

1. En termes d'attributs.
2. Paire de MCDs.

MCDs et la génération des réécritures, la troisième étape consiste à déterminer, pour chaque réécriture rw, les variables qui doivent être des variables de tête de rw. Premièrement, pour chaque variable de tête x dans la requête q telle que $\varphi_C(x)$ est une variable de tête dans $h_C(v)$, x est une variable de tête de rw. Par la suite, toute variable y qui est nouvellement introduite par les vues utilisées dans la génération de rw, y est une variable de tête de rw. Plus précisément, une variable y est dite nouvellement introduite par la vue v_i si $y \in headvars(h_C(v_i))$ et $y \in vars(g')$ où $g' \in atomes(h_C(v_i))$ tel que g' n'est unifié à aucun atome g de q. Plus formellement, pour chaque réécriture rw générée par l'algorithme $\mathcal{H}MiniCon$, pour toute variable $y \in vars(rw)$, si $y \notin \varphi(vars(q))$ alors ajouter y comme variable de tête de rw. Cette modification est nécessaire pour assurer la maximalité de l'approche (voir chapitre 5, page 57).

Afin de mieux comprendre nos motivations, nous illustrons dans l'exemple 9 la nécessité d'assouplir la condition sur les variables de tête et dans l'exemple 10 (section 3.4, page 33) la nécessité d'ajouter certaines variables dans la tête des réécritures pour assurer la propriété de maximalité des vues.

Exemple 9 Supposons la requête q qui retourne les informations sur tous les patients. La vue v contient seulement une partie des informations de tous les patients.

$q(x, y) \leftarrow patients(x, y).$
$\mathcal{V} : v(x') \leftarrow patients(x', y').$

En appliquant l'algorithme MiniCon, aucune réécriture de la requête q en utilisant $\mathcal{V}$ ne sera calculée. En effet, la vue v ne sera pas considérée comme vue contributive ; la condition C_1 sur les variables de tête n'est pas satisfaite. L'attribut y apparait dans l'atome *patients* de la requête q et dans sa tête, alors que $\varphi(y)$ n'apparait pas dans la tête de v étant donné que l'atome $patients(x', y')$ de v est unifié à l'atome $patients(x, y)$ de q avec le mapping $\varphi = \{x \rightarrow x', y \rightarrow y'\}$. La conclusion est qu'il n'y a aucun moyen de répondre à la requête q en utilisant $\mathcal{V}$. Mais dans le contexte de notre travail, cette condition devient trop restrictive. La réécriture rw ci-dessous permet, quant à elle, de répondre partiellement à la requête :

$rw(x) \leftarrow patients(x, y).$ ◊

Il est important de noter ici que l'assouplissement de la condition sur les variables de tête n'est pas réalisé pour toutes les variables n'apparaissant pas dans la tête de la vue. En effet, si une variable distinguée est une variable de jointure dans la requête, il est alors important de vérifier qu'elle apparait dans les variables de tête de la vue si celle-ci ne considère pas la jointure dans sa définition. Dans le cas contraire, nous ne pourrons pas considérer la vue comme contributive.

3.4 $\mathcal{H}MiniCon^+$

Dans cette section, nous présentons les algorithmes de notre approche. Nous allons illustrer le déroulement de l'enchainement des réécritures et nous montrerons à travers l'exemple 10 la nécessité de garder dans la tête des réécritures un ensemble de variables dans le but de garantir la maximalité des résultats.

Exemple 10 Soient les deux ensembles de vues définies sur le même schéma :

$\mathcal{V}_1 : v_{11}(x, y) \leftarrow patients(x, y).$
$\quad v_{12}(y, t) \leftarrow traitements(y, z), docteurs(z, t).$
$\mathcal{V}_2 : v_{21}(x', z') \leftarrow patients(x', y'), traitements(y', z').$
$\quad v_{22}(z', t') \leftarrow docteurs(z', t').$

La première étape consiste donc à définir un ensemble de requêtes $\mathcal{Q}$ sur les vues $\mathcal{V}_1$. Soit $\mathcal{Q}$ l'ensemble des vues de départ :

$\mathcal{Q} : q_1(x, y) \leftarrow v_{11}(x, y).$
$\quad q_2(y, t) \leftarrow v_{12}(y, t).$

Dans le but d'appliquer la première réécriture c.-à-d. réécrire chaque q_i de $\mathcal{Q}$ en utilisant les vues $\mathcal{V}_2$. L'algorithme effectue d'abord une expansion des requêtes (remplacer les atomes des requêtes par leur définition). Nous avons donc :

$q_1^{exp}(x, y) \leftarrow patients(x, y).$
$q_2^{exp}(y, t) \leftarrow traitements(y, z_1), docteurs(z_1, t).$

Où z_1 est une variable fraiche introduite lors de l'expansion.

Pour la suite du déroulement de l'algorithme, nous nous intéressons à la réécriture de la requête q_1 (l'algorithme s'applique de la même manière pour toutes les requêtes q_i dans $\mathcal{Q}$). Nous allons dans ce qui suit présenter deux scénarios du déroulement de l'enchainement des réécritures. Le premier est l'application de l'algorithme $\mathcal{H}MiniCon$ sans l'ajout de certaines variables à la tête des réécritures. Le deuxième scénario est l'application de l'algorithme $\mathcal{H}MiniCon$ avec l'ajout des variables à la tête des réécritures. Le but de cette modification est de garantir une maximalité du résultat.

1. $q_1^{exp}(x, y) \leftarrow patients(x, y)$.
2. $rw_1(x) \leftarrow v_{21}(x, z_2)$.
3. $rw_1^{exp}(x) \leftarrow patients(x, y_1), traitements(y_1, z_2)$.
4. $rw_2(x) \leftarrow v_{11}(x, y_1), v_{12}(y_1, t_1)$.
5. $rw_2^{exp}(x) \leftarrow patients(x, y_1), traitements(y_1, z_3), docteurs(z_3, t_1)$.
6. $rw_3(x) \leftarrow v_{21}(x, z_3), v_{22}(z_3, t_1)$.
7. $rw_3^{exp}(x) \leftarrow patients(x, y_2), traitements(y_2, z_3), docteurs(z_3, t_1)$.
8. $rw_4(x) \leftarrow v_{11}(x, y_2), v_{12}(y_2, t_1)$.
9. $rw_4^{exp}(x) \leftarrow patients(x, y_2), traitements(y_2, z_4), docteurs(z_4, t_1)$.

Scénario 1 : Application de $\mathcal{H}$MiniCon sans la troisième étape

1. $q_1^{exp}(x, y) \leftarrow patients(x, y)$.
2. $rw_1(x, z_2) \leftarrow v_{21}(x, z_2)$.
3. $rw_1^{exp}(x, z_2) \leftarrow patients(x, y_1), traitements(y_1, z_2)$.
4. $rw_2(x, t_1) \leftarrow v_{11}(x, y_1), v_{12}(y_1, t_1)$.
5. $rw_2^{exp}(x, t_1) \leftarrow patients(x, y_1), traitements(y_1, z_3), docteurs(z_3, t_1)$.
6. $rw_3(x, t_1) \leftarrow v_{21}(x, z_3), v_{22}(z_3, t_1)$.
7. $rw_3^{exp}(x, t_1) \leftarrow patients(x, y_2), traitements(y_2, z_3), docteurs(z_3, t_1)$.
8. $rw_4(x, t_1) \leftarrow v_{11}(x, y_2), v_{12}(y_2, t_1)$.
9. $rw_4^{exp}(x, t_1) \leftarrow patients(x, y_2), traitements(y_2, z_4), docteurs(z_4, t_1)$.

Scénario 2 : Application de $\mathcal{H}$MiniCon avec la troisième étape

À la première application de l'algorithme $\mathcal{H}MiniCon$, une seule réécriture est calculée (Ligne **2**). Nous remarquons ici que le corps de la réécriture est le

même dans les deux scénarios mais la tête diffère. Dans le deuxième scénario, nous avons ajouté la variable z_2 dans la tête de la réécriture. En effet, nous pouvons considérer la variable z_2 comme nouvellement introduite par la vue (la variable z_2 n'apparait pas dans $vars(q_1)$). En regardant l'expansion de la réécriture (Ligne **3**), la variable z_2 a été introduite par l'atome *traitements* qui est nouvellement introduit par la définition de la vue (l'atome *traitements* n'est unifié a aucun atome de la requête). Dans ce cas, nous ajoutons toutes les variables des nouveaux atomes n'apparaissant pas dans les atomes unifiés, mais apparaissant dans la tête de la vue, à la tête de la réécriture.
De la même manière, l'algorithme réécrira l'expansion de rw_1 en utilisant cette fois l'ensemble $\mathcal{V}_1$. Une attention particulière est à porter à la tête de la réécriture rw_2 (Ligne **4**). Dans le scénario 2, nous remarquons que la variable z_2 n'est plus dans la tête. Ceci est dû au fait que les vues utilisées pour la réécriture ne projettent pas la variable z_2. De la même manière, une nouvelle variable t_1 apparait dans la tête car elle apparait dans l'atome *docteurs* qui est introduit par la vue v_{12} et qui n'est unifié a aucun atome de la requête (rw_1).
Dans le but de déterminer si après l'application de la double réécriture, l'algorithme atteint son objectif, c.-à-d. déterminer les ensembles de tuples qui peuvent être calculés à partir des deux ensembles de vues, nous proposons de vérifier pour toute réécriture rw_i générée par l'algorithme de double réécriture, si rw_i contient la requête de départ. La vérification de cette condition implique que tous les tuples de la requête de départ sont inclus dans la réécriture (après l'application de la double réécriture). Plus précisément, tous les tuples ont été filtrés et par conséquent l'ensemble des tuples de la requête est accessible à partir des deux ensembles de vues. Dans notre exemple, cela revient à déterminer si $q_1 \sqsubseteq rw_2$. Pour ce faire, nous nous appuyons sur un algorithme de subsomption [CGM90]. Nous déduisons dans cette exemple que l'application d'une seule double réécriture n'est pas suffisante pour atteindre notre objectif (rw_2 ne subsume pas la requête q_1). L'algorithme réappliquera donc la double réécriture (Lignes **6** à **9**). À cette étape, la subsomption de la requête par sa réécriture est vérifiée (rw_4 subsume rw_2). L'algorithme s'arrête et retourne la réécriture rw_4 comme étant une vue correcte.

Le résultat obtenu (Ligne **9**) montre bien la nécessité de garder dans la

tête des réécritures les variables qui apparaissent tout au long du processus de réécriture. En effet, nous remarquons bien que dans le scénario 1, seule la variable x est retournée comme résultat contrairement au scénario 2 qui retourne les variables x et t_1. Ce deuxième cas représente bien l'ensemble des tuples qui peuvent être accessible à partir des deux ensembles de vues. $\Diamond$

Il est nécessaire de préciser que l'application répétée de la double réécriture permet de s'assurer que les variables ajoutées au fur et à mesure dans la tête des réécritures sont accessibles par les deux ensembles de vues. Si une des variables ajoutées n'est pas accessible par un des ensembles de vues, elle ne sera pas retournée lors de l'application de la réécriture en utilisant cet ensemble de vues.

Plus généralement, l'algorithme prend en entrée deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$. Pour la première itération, un ensemble de requêtes $\mathcal{Q}$ est calculé. Ce dernier est défini sur un des ensembles de vues, par exemple $\mathcal{V}_1$. L'algorithme commence par réécrire chaque q_i de $\mathcal{Q}$ en utilisant l'ensemble des vues $\mathcal{V}_2$. Un ensemble de réécritures est obtenu, noté $\mathcal{RW}_{q_i}$. Cette première étape détermine quel ensemble de tuples de q_i est calculable à partir de $\mathcal{V}_1$. La deuxième étape consiste à vérifier si cet ensemble de tuples est aussi calculable à partir de $\mathcal{V}_2$. Pour cela, l'algorithme réécrit chaque rw_j de $\mathcal{RW}_{q_i}$ en utilisant l'ensemble des vues $\mathcal{V}_1$. Nous notons $\mathcal{RW}_{rw_j}$ l'ensemble des réécritures générées par cette deuxième réécriture. Cette double réécriture est effectuée par l'algorithme 1.

Algorithme 1 : *Algorithme de double réécriture*

Données : *q : La requête à réécrire*
$\mathcal{V}_1$, $\mathcal{V}_2$: Deux ensembles de vues définies sur le même schéma
Résultat : *$\mathcal{RW}$: Ensemble de réécritures*

q^{exp} = *expansion(q)*
$\mathcal{RW}_q$ = $\mathcal{H}$*MiniCon(q^{exp},$\mathcal{V}_2$)*
pour *chaque réécriture rw_j de $\mathcal{RW}_q$* **faire**
  rw_j^{exp} = *expansion(rw_j)*
  $\mathcal{RW}_{rw_j}$ = $\mathcal{H}$*MiniCon(rw_j^{exp},$\mathcal{V}_1$)*
  Ajouter $\mathcal{RW}_{rw_j}$ à $\mathcal{RW}$
Retourner $\mathcal{RW}$;

Après cette étape, l'algorithme doit vérifier si l'application de la double réécriture en utilisant les deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$ a filtré des tuples de $\mathcal{Q}$. Pour cela, nous proposons de vérifier pour chaque requête q_i si chacune de ses réécritures contient q_i en utilisant l'algorithme de subsomption. Nous rappelons ici que l'application de l'algorithme $\mathcal{H}$MiniCon produira des réécritures qui n'ont pas nécessairement le même schéma [3]. Donc pour vérifier l'inclusion, l'algorithme sélectionnera seulement les réécritures comparables $\mathcal{CRW}$ et vérifiera pour chaque réécriture rw_j dans $\mathcal{CRW}$ si $q_i \sqsubseteq rw_j$. Une réécriture comparable [WMT02] est une réécriture qui a le même schéma que la requête. Pour toutes les réécritures qui subsument la requête, cela implique qu'aucun des tuples de q_i n'a été filtré par l'application de la double réécriture. Autrement dit, tous les tuples de q_i sont calculables à partir des deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$. Donc, l'algorithme termine et retourne ces réécritures comme nouvelles vues. Sinon, dans le cas où les réécritures ne sont pas comparables à la requête ou ne la subsument pas, ceci implique qu'une des deux étapes de réécriture a filtré certains tuples. Dans ce cas, l'algorithme de double réécriture est réappliqué en considérant cette fois-ci ces réécritures comme des requêtes à réécrire. Cette itération de double réécriture est effectuée par l'algorithme 2.

Les vues générées par notre algorithme sont définies sur l'ensemble $\mathcal{V}_1$. Il suffit de réaliser l'expansion de chaque vue de $\mathcal{W}$ pour obtenir une définition sur les relations de base.

3. Nous avons relaxé la condition sur les variables de tête et ajouté certaines variables.

Algorithme 2 : *Algorithme $\mathcal{H}MiniCon^+$*

Données : $\mathcal{V}_1$, $\mathcal{V}_2$ *: Deux ensembles de vues définies sur le même schéma*

Résultat : $\mathcal{W}$ *: Ensemble de vues définies sur* $\mathcal{V}_1$

Définir $\mathcal{Q}$ *: Ensemble de requêtes de départ définies sur* $\mathcal{V}_1$

tant que $\mathcal{Q}$ *est non vide* **faire**

- *prendre* q_i *de* $\mathcal{Q}$
- $\mathcal{RW}_{q_i}$ = *Double_ Réecritures*(q_i, $\mathcal{V}_1$, $\mathcal{V}_2$)
- $\mathcal{CRW}_{q_i} = \{rw | rw \in \mathcal{RW}_{q_i}$ *et* $headvars(rw) = headvars(q_i)\}$
- **pour** *chaque réécriture* rw_k *de* $\mathcal{RW}_{q_i}$ **faire**
  - **si** rw_k *est dans* $\mathcal{CRW}_{q_i}$ **alors**
    - **si** rw_k *subsume* q_i **alors**
      - *Ajouter* rw_k *à* $\mathcal{W}$
    - **sinon**
      - *Ajouter* rw_k *à* $\mathcal{Q}$
  - **sinon**
    - *Ajouter* rw_k *à* $\mathcal{Q}$

Retourner $\mathcal{W}$;

3.4.1 Arbre de réécritures

Dans cette section, nous définissons deux structures qui résultent de l'application de l'algorithme $\mathcal{H}MiniCon^+$. La première est l'arbre de réécritures. À chaque requête dans $\mathcal{Q}$ est associé un arbre de réécritures $\mathcal{RT}$.

Définition 3.3 (Arbre de réécritures $\mathcal{RT}$)**.** *Soient q une requête à réécrire, $\mathcal{V}_1$ et $\mathcal{V}_2$ deux ensembles de vues. L'arbre de réécritures associé à q, noté $\mathcal{RT}(q)$ ou simplement $\mathcal{RT}$, est défini comme suit :*

- *La racine est la requête q.*
- *Les noeuds de profondeur $k+1$ sont les réécritures générées par l'algorithme $\mathcal{H}MiniCon$ en réécrivant les noeuds de profondeur k en utilisant soit l'ensemble $\mathcal{V}_1$ soit l'ensemble $\mathcal{V}_2$. Un noeud n_{k+1} est l'enfant du noeuds n_k si le noeud n_{k+1} est une réécriture de la requête n_k.*

Exemple 11 La figure 3.1 représente un exemple d'arbre de réécritures :

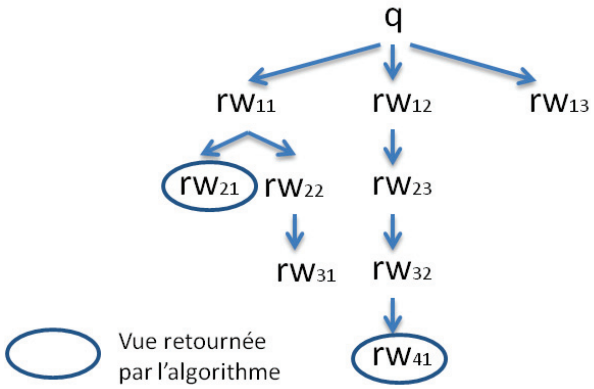

FIGURE 3.1 – Arbre de réécritures.

Nous remarquons que les vues retournées par l'algorithme sont quelques feuilles de l'arbre. ◊

Dans le but d'optimiser le processus de réécriture, nous remplaçons au fur et à mesure de la génération de l'arbre (par l'algorithme $\mathcal{H}MiniCon^{+}$) certaines réécritures par des requêtes équivalentes. Cette modification n'impacte pas le résultat retourné par l'approche. En effet, réécrire une requête q ou réécrire sa requête équivalente q' en utilisant le même ensemble de vues, retourne le même ensemble de tuples. Nous verrons dans le chapitre 4 (page 43) que cette modification est nécessaire pour démontrer la terminaison de l'approche sous certaines conditions.

3.4.2 Arbre d'atomes

À partir de l'arbre de réécritures associé à chaque requête, nous définissons des arbres d'atomes. Ces arbres contiennent les informations concernant la réécriture de chaque atome (la vue qui l'a généré, l'atome de la requête auquel il a été unifié, etc.). À partir de ces informations, il est possible de caractériser et de détecter l'apparition de boucles dans le processus de réécriture. Cette caractérisation faite, il est possible de définir sous quelles conditions notre approche termine.

Définition 3.4 (Arbre d'atomes $\mathcal{AT}$). *Étant donnée une branche $\mathcal{X} = B^0, B^1, ...$ de l'arbre de réécritures $\mathcal{RT}$, l'arbre d'atomes $\mathcal{AT}(\mathcal{RT})$, ou simplement $\mathcal{AT}$,*

de $\mathcal{RT}$, est défini comme suit :

- *La racine est un noeud anonyme r.*
- *Les noeuds de profondeur $K+1$ sont les enfants des atomes de B^k, notés g^k où $k \geq 1$.*
- *g'^{k+1} est l'enfant de g^k de type :*
  - *Direct : S'il existe un MCD C de la réécriture B^{k+1} tel que : $\varphi_C(g^k) = g'^{k+1}$.*
  - *Indirect : Si g'^{k+1} appartient à l'expansion de la vue v utilisée pour réécrire g^k et g'^{k+1} n'a pas de parent direct.*

Étant donné que dans l'algorithme MiniCon une vue peut réécrire plus d'un atome à la fois, un noeud g'^{k+1} peut avoir plusieurs parents indirects. Dans ce cas, et dans un souci de rester dans une configuration d'arbres, nous choisissons arbitrairement un seul parent indirect de g'^{k+1} à partir de G_C, où G_C est l'ensemble des atomes couverts par la vue v.

On note :

- view$(g'^{k+1}) = v$;
- cpos(g'^{k+1}) la position de l'atome correspondant à g'^{k+1} dans v ;
- ppos(g'^{k+1}) la position de l'atome correspondant à g'^{k} dans v ;
- type$(g'^{k+1}) = Direct$ or $Indirect$

On note $g \xrightarrow{\text{dir.}} g'$ (resp. $g \xrightarrow{\text{ind.}} g'$) Si $parent(g') = g$ et type$(g') = Direct$ (resp. type$(g') = Indirect$) et $\xrightarrow{\text{dir.}}{}^*$ la fermeture transitive et réflexive de $\xrightarrow{\text{dir.}}$.

Exemple 12 La figure 3.2 montre une partie de l'arbre d'atomes de la seule branche de l'arbre de réécritures de l'exemple 10 (page 33).

◊

3.5 Correction des vues

Nous démontrons dans cette section que quel que soit l'ensemble de tuples retourné par $\mathcal{W}$ [4], cet ensemble est accessible par les deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$. La propriété suivante garantit la correction des vues.

4. L'ensemble des vues générées par l'algorithme $\mathcal{H}MiniCon^+$.

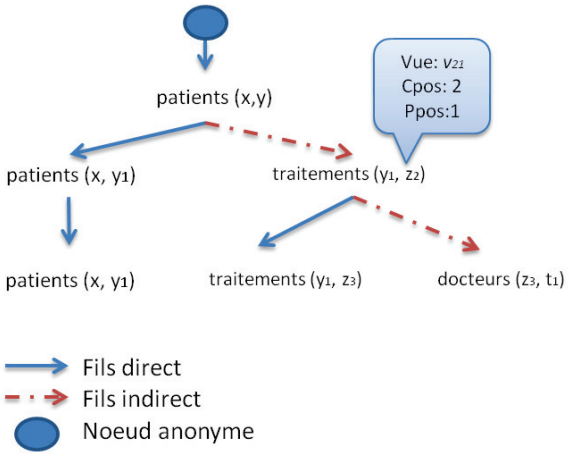

FIGURE 3.2 – Arbre d'atomes de la branche de $\mathcal{RT}(q_1)$ de l'exemple 10.

Propriété 3.5 (Sécurité des vues)**.** *Soient $\mathcal{W}$ l'ensemble des vues générées par l'algorithme $\mathcal{H}MiniCon^+$ et $q^{\mathcal{W}}$ une requête définie sur $\mathcal{W}$. Il existe alors une requête $q^{\mathcal{V}_1}$ définie sur $\mathcal{V}_1$ et une requête $q^{\mathcal{V}_2}$ définie sur $\mathcal{V}_2$ telles que $q^{\mathcal{W}} \equiv q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2}$.*

Démonstration. Soit gv_i une vue générée par l'algorithme $\mathcal{H}MiniCon^+$. Par définition de la condition d'ajout d'une requête dans $\mathcal{W}$ de $\mathcal{H}MiniCon^+$, nous avons :

$$rw^{i,l-1} \equiv rw^{i,l} \equiv gv_i \tag{3.1}$$

où $rw^{i,l}$ est la réécriture de profondeur l de la branche i de l'arbre de réécritures, obtenue par l'application de $\mathcal{H}MiniCon(rw^{i,l-1}, \mathcal{V}_2)$ et gv_i est une des réécritures obtenue par l'application de $\mathcal{H}MiniCon(rw^{i,l}, \mathcal{V}_1)$. La réécriture $rw^{i,l-1}$ est définie en termes de $\mathcal{V}_1$ et la réécriture $rw^{i,l}$ est définie en termes de $\mathcal{V}_2$.

Nous devons maintenant montrer que pour chaque requête $q^{\mathcal{W}}$ sur $\mathcal{W}$, il existe une requête équivalente $q^{\mathcal{V}_1}$ sur $\mathcal{V}_1$ et une requête équivalente $q^{\mathcal{V}_2}$ sur $\mathcal{V}_2$.
Considérons la requête suivante sur $\mathcal{W}$:

$$q^{\mathcal{W}} = q(\bar{X}) \leftarrow gv_1(\bar{X}_1), ..., gv_n(\bar{X}_n).$$

Nous définissons $q^{\mathcal{V}_1}$ comme suit : $q^{\mathcal{V}_1} = expansion(q^{\mathcal{W}})$.
Nous rappelons que les vues générées sont par définition définies sur $\mathcal{V}_1$ et par

conséquent nous avons :

$$q^{\mathcal{W}} \equiv q^{\mathcal{V}_1} \tag{3.2}$$

où $q^{\mathcal{V}_1}$ est une requête sur $\mathcal{V}_1$.

À partir de (3.1) et pour chaque $gv_i(\bar{X}_i)$, nous avons $gv_i(\bar{X}_i) \equiv rw_i^{i,l}(\bar{X}_i)$. Nous définissons alors q' comme suit :

$$q' = q'(\bar{X}) \leftarrow rw_1^{1,l}(\bar{X}_1), ..., rw_n^{n,l}(\bar{X}_n)$$

Nous avons :

$$q^{\mathcal{W}} \equiv q' \tag{3.3}$$

Nous définissons $q^{\mathcal{V}_2}$ comme suit : $q^{\mathcal{V}_2} = expansion(q')$. En effet, nous rappelons que les réécritures $rw^{i,l}$ sont définies en termes de $\mathcal{V}_2$.
Nous avons :

$$q' \equiv q^{\mathcal{V}_2} \tag{3.4}$$

À partir de (3.3) et (3.4), nous concluons que :

$$q^{\mathcal{W}} \equiv q^{\mathcal{V}_2} \tag{3.5}$$

où $q^{\mathcal{V}_2}$ est une requête sur $\mathcal{V}_2$.

À partir de (3.2) et (3.5), nous concluons que les vues générées par l'algorithme $\mathcal{H}MiniCon^+$ sont correctes. □

Chapitre 4

Terminaison de l'algorithme $\mathcal{H}$MiniCon$^+$

Sommaire

4.1 Introduction

Nous traitons dans ce chapitre la terminaison de notre algorithme. Nous caractérisons sous quelles conditions l'algorithme $\mathcal{H}MiniCon^+$ termine, c.-à-d. l'enchainement de la double réécriture atteint toujours un point fixe. Nous caractérisons également les cas où il est impossible de déterminer sa finitude. En effet, nous verrons que dans des cas qui dépendent de la définition des vues données en entrée ($\mathcal{V}_1$ et $\mathcal{V}_2$), certaines branches des arbres de réécritures peuvent être infinies ; une boucle se crée dans le processus d'enchainement de réécritures. Nous illustrons ceci à travers deux exemples et présentons par la suite comment il est possible de détecter ces boucles et montrons qu'excepté ce cas l'algorithme $\mathcal{H}MiniCon^+$ termine.

Exemple 13 Considérons l'ensemble des vues suivant :

$\mathcal{V}_1$: $v_{11}(x, y) \leftarrow r_1(x, y)$.
$v_{12}(x, z) \leftarrow r_2(x, y), r_1(y, z)$.

$\mathcal{V}_2 : v_{21}(x, y) \leftarrow r_1(x, y), r_2(y, z).$
$v_{22}(x, y) \leftarrow r_2(x, y).$

La figure 4.1 présente l'arbre d'atomes d'une des branches de l'arbre de réécritures de la requête q_0 telle que :

$expansion(q_0) : q_0^{exp}(x, y) \leftarrow r_1(x, y)$

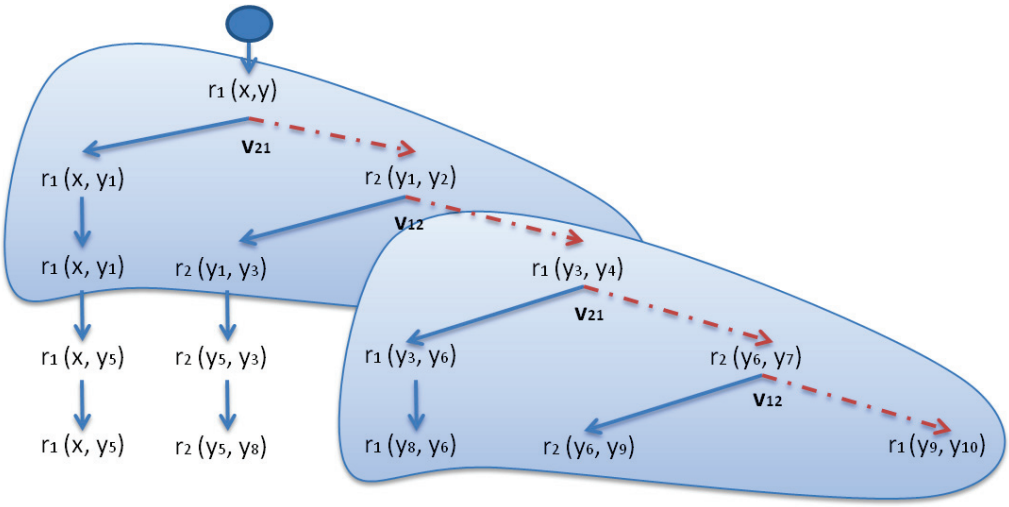

FIGURE 4.1 – Arbre d'atomes de la branche de $\mathcal{RT}(q_0)$ qui boucle.

Dans cet exemple, l'algorithme réappliquera à l'infini la double réécriture. Ceci se produit à cause d'une boucle dans le processus de réécriture. À chaque fois que l'algorithme réécrit l'atome r_1, le processus de double réécriture utilise le même sous ensemble de vues $(v_{21},\ v_{12})$ dans une de ses réécritures et par conséquent génère le même ensemble d'atomes : r_1, r_2, r_1. Nous remarquons ici que la réécriture de l'atome r_1 génère un atome additionnel r_1, ce qui crée une certaine forme de boucle. Dans ce cas, cette réécriture ne peut vérifier la condition d'arrêt de l'algorithme et par conséquent ne terminera jamais son exécution car, de la même manière, l'algorithme utilisera le même sous ensemble de vues $(v_{21},\ v_{12})$ pour réécrire l'atome additionnel r_1 généré auparavant. ◊

L'exemple suivant nous montre que dans certains cas les réécritures générées contiennent toujours plus d'atomes que les requêtes et par conséquent, risquent de ne jamais satisfaire le test de subsomption, ce qui empêchera l'al-

gorithme de s'arrêter.

Exemple 14 Considérons l'ensemble des vues suivant :

$\mathcal{V}_1 : v_{11}(x,y) \leftarrow r_1(x,y), r_3(y,z).$
$v_{12}(x,y) \leftarrow r_2(x,y).$
$v_{13}(x,y) \leftarrow r_3(x,y).$
$\mathcal{V}_2 : v_{21}(x,y) \leftarrow r_1(x,y), r_2(y,z).$
$v_{22}(x,y) \leftarrow r_2(x,y).$
$v_{23}(x,y) \leftarrow r_3(x,y).$

La figure 4.2 présente l'arbre d'atomes d'une des branches de l'arbre de réécritures de la requête q_0 telle que :

$expansion(q_0) : q_0^{exp}(x,y) \leftarrow r_1(x,y), r_3(y,z_1).$

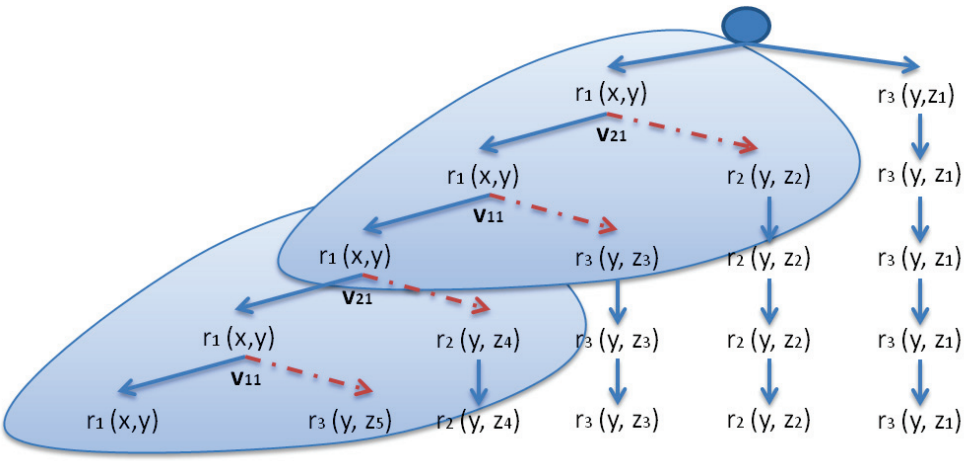

FIGURE 4.2 – Arbre d'atomes d'une branche de $\mathcal{RT}(q_0)$.

Nous remarquons dans cette branche que l'algorithme utilise le même couple de vues (v_{21}, v_{11}) à chaque réécriture de l'atome r_1. Par la suite, chaque atome supplémentaire [1] de cette réécriture est réécrit indépendamment par les autres vues $(v_{12}, v_{13}, v_{22}, v_{23})$. Par conséquent, il y aura toujours des atomes supplémentaires qui s'ajouteront lors de la réécriture de r_1 et peuvent faire étendre la requête indéfiniment. ◊

Nous montrons dans la suite de ce chapitre que dans le cas de l'exemple 14,

1. L'atome généré par un arc indirect lors de la réécriture.

il est possible de caractériser certains atomes qui s'ajoutent à chaque étape. Nous prouverons que la requête qui ne contient pas ces atomes est équivalente à la requête d'origine et comme définit dans l'arbre de réécritures (page 38), nous nous permettons de remplacer une requête par une requête équivalente. Ceci nous permettra d'éviter de réécrire inutilement des parties de la requête. Afin de caractériser ces atomes, nous présentons comment définir pour chaque noeud son historique et à partir de son type (Direct ou Indirect) déterminer si c'est un noeud réel ou virtuel.

Concernant l'exemple 13, cette configuration se réalise quand il y a une certaine forme d'interactions entre les deux ensembles de vues. Nous proposons dans ce cas de signaler cette interaction à l'utilisateur par un message lui spécifiant exactement les vues concernées par le processus de boucle générée. Ceci lui permettra de modifier les définitions des vues si cela est possible. Dans le cas où il est impossible de modifier la définition des vues données en entrée, l'approche ne garantit plus la maximalité des résultats. En effet, ne pouvant pas dérouler l'algorithme indéfiniment, l'algorithme arrêtera la branche à la rencontre de ce cas de boucle, et par conséquent les futures réécritures de cette branche qui satisfont la condition d'arrêt ne seront pas calculées.

4.2 Caractéristiques des noeuds

Avant d'introduire la notion d'historique d'un noeud, nous définissons la fonction de partitionnement des variables d'un atome. Cette fonction nous permet de déterminer les unifications faites dans chaque atome.

Définition 4.1 (Fonction de partitionnement)**.** *Soit g un noeud dans $\mathcal{AT}$. $Part_g$ est la fonction de partitionnement de $vars(g)$ définie comme suit :*

$$Part_g = \{\{j \mid x_i = x_j \text{ où } x_i, x_j \in vars(g) \text{ et } 1 \leq j \leq n\} \mid 1 \leq i \leq n\}$$

Exemple 15 Soit l'atome $r_1(x_1, x_2, x_1)$. $Part_{r_1}$ est la liste suivante : $Part_{r_1} = \{\{1, 3\}, \{2\}\}$ ◊

Remarque : Il est important de noter que pour un atome donné, le nombre des différents partitionnements possibles des variables est fini.

4.2.1 Historique d'un noeud

Nous définissons dans cette section comment calculer l'historique de chaque noeud en prenant en compte les informations concernant l'atome père et la vue utilisée pour le réécrire. L'historique du noeud consiste à garder trace des informations des atomes ancêtres qui ont été générés de manière indirecte (atomes supplémentaires) et les unifications qui ont été faites tout au long du processus de réécriture.

Définition 4.2 (Historique)**.** *Pour chaque noeud g dans $\mathcal{AT}$ excepté pour la racine, $History(g)$ est une liste définie comme suit :*

- *Si g est un enfant de la racine, alors $History(g) = [(pos, Part_g)]$ où pos est la position de g dans la requête ;*
- *Si $type(g) = Indirect$ alors :*
  $History(g) = History(parent(g)) + [(H_g, Part_g)]$
  où $H_g = (view(g), cpos(g), ppos(g))$
- *Si $type(g) = Direct$ alors :*
  - *Si $Part_g = Part_{parent(g)}$ alors : $History(g) = History(parent(g))$*
  - *Sinon : Soit $History((parent(g)) = [L, (H_{parent(g)}, Part_{parent(g)})]$*
    alors $History(g) = [L, (H_{parent(g)}, Part_g)]$

Exemple 16 Reprenons l'exemple 13 (page 43). Nous complétons l'arbre par l'historique des noeuds dans la figure 4.3.

L'historique de l'atome $r_1(x, y)$ (noeud fils de la racine) est défini à partir de (1) la position de l'atome r_1 dans la requête de départ et (2) la liste des unifications faites au niveau de l'atome. Nous remarquons que l'atome $r_1(x, y_5)$ a le même historique que l'atome de départ. En effet, tout au long du processus de réécriture, tous les atomes ont été générés directement, c.-à-d. aucun des atomes pères n'a été un atome supplémentaire généré lors de la réécriture d'un autre atome et aucune unification supplémentaire n'a été réalisée par l'ensemble des vues utilisées dans les différentes réécritures. Nous attirons l'attention sur l'atome $r_2(y_6, y_7)$ qui a comme historique $[(1, \{\{1\}, \{2\}\}), ([v_{21}, 1, 2], \{\{1\}, \{2\}\}), ([v_{12}, 1, 2], \{\{1\}, \{2\}\}), ([v_{21}, 1, 2], \{\{1\}, \{2\}\})]$. Nous concluons de cet historique et plus précisément de l'apparition double du couple $([v_{21}, 1, 2], \{\{1\}, \{2\}\})$ qu'il y a une boucle dans le processus de réécriture. Dans ce cas, nous proposons d'arrêter le processus de réécriture de

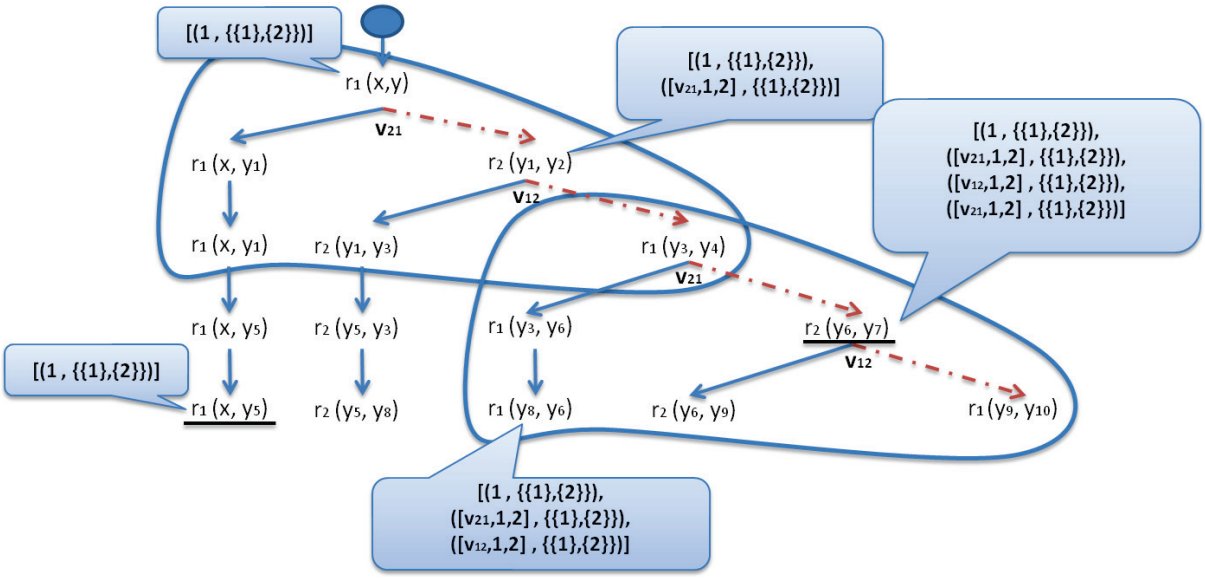

FIGURE 4.3 – Historique des atomes.

cette branche et d'en tenir informé l'utilisateur qu'il y a une forme d'interaction entre les vues v_{21} et v_{12}. ◊

À partir de là, nous démontrons dans la suite que la condition nécessaire pour que l'algorithme $\mathcal{H}MiniCon^+$ termine est que tous les atomes dans les arbres d'atomes de toutes les branches de l'arbre de réécritures n'ont pas dans leur historique deux fois le même couple. Ceci est défini dans le théorème suivant.

Théorème 4.3 (Terminaison sous conditions). *Étant donnés une requête q et deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$, si pour chaque branche $\mathcal{X}$ de l'arbre généré $\mathcal{RT}$ par $\mathcal{H}MiniCon^+(q, \mathcal{V}_1, \mathcal{V}_2)$ et pour chaque noeud g de l'arbre d'atomes $\mathcal{AT}$ de $\mathcal{X}$, $History(g)$ ne contient pas deux fois le même couple, alors $\mathcal{RT}$ est fini.*

Dans le but de prouver le théorème 4.3, nous introduisons une série de définitions et de lemmes intermédiaires.

4.2.2 Noeuds réels et virtuels

À partir de l'historique de chaque noeud et de son type (Direct ou Indirect) nous caractérisons si le noeud est *réel* ou *virtuel*. Cette caractérisation nous

permet de traiter les cas de boucles présentés dans l'exemple 14 (page 45).

Définition 4.4 (Noeuds réels et virtuels)**.** *Soient une requête q, deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$, $\mathcal{X}$ une branche de l'arbre de réécritures $\mathcal{RT}(q)$ généré par $\mathcal{H}MiniCon^+(q, \mathcal{V}_1, \mathcal{V}_2)$, et $\mathcal{AT}(\mathcal{X})$ l'arbre d'atomes de la branche $\mathcal{X}$. Un noeud g_i^k est dit noeud virtuel s'il y'a un noeud g_j^k où $History(g_i^k) = History(g_j^k)$ et $type(g_i^k) = Indirect$, k étant la profondeur du noeud dans l'arbre $\mathcal{AT}$. Un noeud g_i^k est dit noeud réel s'il n'est pas virtuel.*

Exemple 17 La figure 4.4 illustre les noeuds réels et virtuels de l'exemple 14. Nous remarquons que les atomes $r_3(y, z_5)$ et $r_3(y, z_3)$ ont le même historique

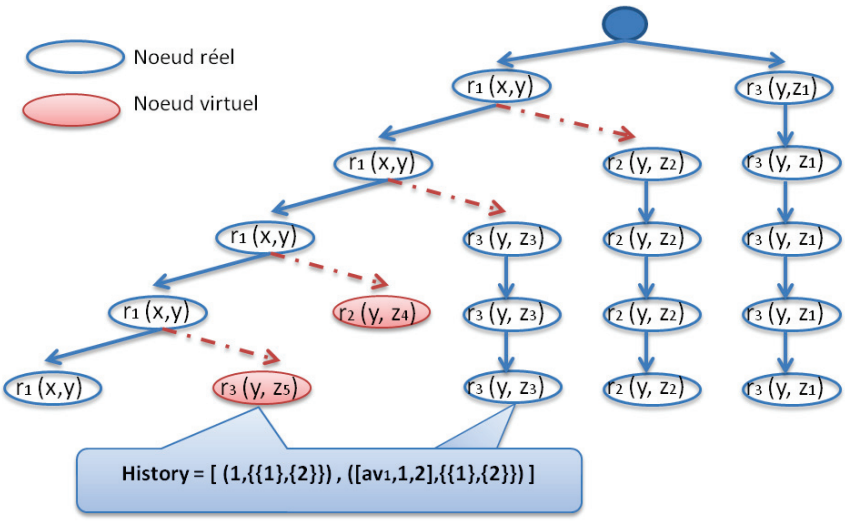

FIGURE 4.4 – Noeuds réels et virtuels.

et que l'atome $r_3(y, z_5)$ a été généré de manière indirecte. Par conséquent, l'atome $r_3(y, z_5)$ est un noeud virtuel. ◊

4.3 Arbres de réécritures et d'atomes réels

Nous rappelons qu'à chaque noeud (réécriture) rw^l de l'arbre de réécritures $\mathcal{RT}(q_i)$ correspond un ensemble de noeuds (atomes) g_i^l de l'arbre d'atomes $\mathcal{AT}(\mathcal{X})$ où rw^l est un noeud de la branche $\mathcal{X}$. Dans cette section, nous démontrons que la requête résultant de l'élimination des atomes correspondants

aux noeuds virtuels est équivalente à la requête d'origine. Nous définissons pour cela l'arbre de réécritures et d'atomes réels et leurs propriétés.

Définition 4.5 (Arbres de réécritures et d'atomes réels)**.** *Soient une branche $\mathcal{X}$ de l'arbre de réécritures, $\mathcal{AT}(\mathcal{X})$ l'arbre d'atomes de $\mathcal{X}$ et un élément rw^l de $\mathcal{X}$. Chaque atome dans rw^l est associé à un noeud dans $\mathcal{AT}(\mathcal{X})$. On définit $real(\mathcal{AT}(\mathcal{X})) = \mathcal{AT}(\mathcal{X}) - G_V$ où G_V est l'ensemble des noeuds virtuels. Chaque atome dans $real(rw^l)$ est associé à un atome g_i^l de $real(\mathcal{AT}(\mathcal{X}))$.*

Propriété 4.6 (Caractérisation des réécritures réelles)**.** *Étant donnés une branche $\mathcal{X}$ de l'arbre de réécritures $\mathcal{RT}(q_i)$ et un élément rw^l dans $\mathcal{X}$, $rw^l \equiv real(rw^l)$.*

Afin de démontrer que $rw^l \sqsubseteq real(rw^l)$, nous définissons un mapping γ de rw^l vers $real(rw^l)$ et démontrons que c'est un morphisme. Pour cela, nous introduisons différents mappings présents dans l'arbre de réécritures et l'arbre d'atomes.

Définition 4.7 (Mapping de rw^k vers rw^{k+1})**.** *Des propriétés de l'algorithme de réécriture, nous définissons un mapping δ^k de rw^k (la requête) vers rw^{k+1} (la réécriture) tel que $rw^{k+1} \Rightarrow \delta^k(rw^k)$. Pour chaque atome g^k dans rw^k, il existe un atome g^{k+1} dans rw^{k+1} où $g^k \xrightarrow{dir.} g^{k+1}$ tel que $g^{k+1} = \delta^k(g^k)$ avec δ^k étant un mapping qui unifie les variables de g^k aux variables de g^{k+1} comme suit :*

- *$\delta^k(x) = x$ Si $x \in headvars(\sigma(v))$ où $view(g^{k+1}) = v$ et σ est le mapping d'expansion de v.*
- *$\delta^k(x) = y$ Si $\sigma(x) = y$ et $y \in headvars(\sigma(v))$ où $view(g^{k+1}) = v$ et σ est le mapping d'expansion de v.*
- *$\delta^k(x) = y$ où y est une variable fraiche, sinon.*

Définition 4.8 (Mapping de rw^k vers rw^l)**.** *Soient $\mathcal{X}$ une branche de l'arbre de réécritures, rw^k et rw^l deux réécritures dans $\mathcal{X}$ où $k < l$. De la définition 4.7 nous pouvons déduire qu'il existe un mapping $\delta^{k\to l}$ de rw^k vers rw^l où $\delta^{k\to l}(rw^k) = \delta^{l-1}$ o δ^{l-2} o ... o $\delta^k(rw^k)$ tel que $rw^l \Rightarrow \delta^{k\to l}(rw^k)$.*

Soient g^l un atome dans rw^l et g^k un atome dans rw^k où $type(g^i) = Direct$ pour tout $k < i \leq l$. Nous avons $g^l = \delta^{k\to l}(g^k)$.

Définition 4.9 (Mapping de g^l vers g^k)**.** *Soit g^l un atome correspondant à un noeud virtuel dans rw^l. Par définition, il existe un atome g'^l dans rw^l avec $History(g^l) = History(g'^l)$ et $type(g'^l) = Direct$. Il existe un certain atome g^k dans rw^k où $k < l$ et $g^k \xrightarrow{dir.}{}^{*} g'^l$ tel que $History(g^k) = History(g'^l)$ et $type(g^k) = Indirect$. Nous avons alors $History(g^k) = History(g^l)$ et $type(g^k) = type(g^l) = Indirect$. Par définition, cela implique que $view(g^l) = view(g^k) = v$.*
Nous définissons un mapping $\delta_{g^l}^{l\to k}$ de $vars(g^l)$ vers $vars(g^k)$ tel que $\delta_{g^l}^{l\to k}(g^l) = g^k$ comme suit :

- *Si x n'est pas une variable fraiche dans v^l alors $\delta_{g^l}^{l\to k}(x) = x$.*
- *Sinon, ceci implique que la variable x a été introduite comme une variable fraiche dans l'étape d'expansion de la vue v et $\delta_{g^l}^{l\to k}(x) = y$ où y est la variable fraiche dans g^k introduite de la même manière dans l'étape d'expansion de la vue v.*

Le lemme suivant démontre que si x est une variable fraiche qui apparait dans plusieurs atomes correspondants à des noeuds virtuels dans rw^l alors les mappings $\delta_{g_i^l}^{l\to k}$ unifient la variable x avec la même variable dans rw^k.

Lemme 4.10. *Si x est une variable fraiche dans rw^l et x apparait dans deux atomes distincts g^l et g'^l qui correspondent à des noeuds virtuels alors $\delta_{g^l}^{l\to k}(x) = \delta_{g'^l}^{l\to k}(x)$.*

Démonstration. Soit x une variable fraiche dans rw^l et x apparait dans deux atomes distincts g^l et g'^l qui correspondent à deux noeuds virtuels. Ceci implique que la variable x a été introduite comme une variable fraiche à l'étape d'expansion de la vue. Donc, g^l et g'^l ont été générés par la même vue v. À partir de la définition 4.9, on en déduit qu'il existe un mapping $\delta_{g^l}^{l\to k}$ et $\delta_{g'^l}^{l\to k}$ tel que $\delta_{g^l}^{l\to k}(g^l) = g^k$ et $\delta_{g'^l}^{l\to k}(g'^l) = g'^k$.

Comme la variable x a été introduite comme une variable fraiche à l'étape d'expansion de v et par définition g^k et g'^k ont été introduits par la même vue, nous concluons que $\delta_{g^l}^{l\to k}(x) = \delta_{g'^l}^{l\to k}(x)$. □

Définition 4.11 (Mapping γ)**.** *Soit g un atome dans rw^l, le mapping γ est défini de $vars(rw^l)$ vers $vars(real(rw^l))$ comme suit :*

- *Si x n'est pas une variable fraiche ou s'il existe un atome g^l qui correspond à un noeud réel tel que $x \in vars(g^l)$ alors $\gamma(x) = x$.*

– *Sinon,* $x \in vars(g^l)$ *pour un certain atome* g^l *correspondant à un noeud virtuel et on définit* $\gamma(x) = \delta^{k \to l}(\delta_{g^l}^{l \to k}(x))$

Exemple 18 Nous illustrons dans la figure 4.5, les différents mappings définis plus haut dans l'exemple 14.

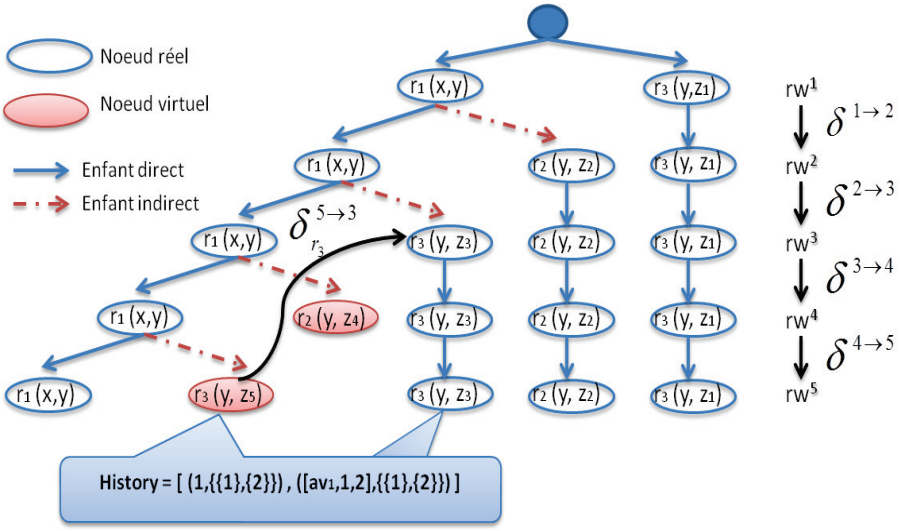

FIGURE 4.5 – Les mappings définis dans un arbre d'atomes.

◊

Démonstration de la propriété 4.6 Afin de démontrer ce théorème, il suffit de prouver que : $rw^l \sqsubseteq real(rw^l)$ et $real(rw^l) \sqsubseteq rw^l$.

$rw^l \sqsubseteq real(rw^l)$: Étant donné que le mapping d'identité est un morphisme de $real(rw^l)$ vers rw^l, il est évident que

$$rw^l \sqsubseteq real(rw^l). \tag{4.1}$$

$real(rw^l) \sqsubseteq rw^l$: Soit g^l un atome dans rw^l correspondant à un noeud réel, alors $\gamma(g^l) = g^l \in real(rw^l)$.
Soit g^l un atome dans rw^l correspondant à un noeud virtuel. De la définition 4.9, il s'ensuit que nous avons un mapping de $vars(g^l)$ vers $vars(g^k)$ où g^k est un certain atome dans rw^k et $k < l$, tel que $\delta_{g^l}^{l \to k}(g^l) = g^k$. (a)
De la définition 4.8, il s'ensuit que nous avons un mapping de $vars(g^k)$ vers $vars(g'^l)$ tel que $\delta^{k \to l}(g^k) = g'^l$. (b)

Puisque type(g'^l) = $Direct$ alors g'^l correspond à un noeud réel. Nous pouvons conclure que pour chaque atome g^l qui correspond à un noeud virtuel, il peut être unifié à un certain atome g'^l dans $real(rw^l)$. À partir du lemme 4.10, nous déduisons que $\gamma(g^l) = \delta^{k\to l}(\delta^{l\to k}_{g^l}(g^l))$.

De (a) et (b), nous avons $\delta^{k\to l}(\delta^{l\to k}_{g^l}(g^l)) = g'^l \in real(rw^l)$. Par conséquent, $\gamma(g^l) \in real(rw^l)$. Il s'ensuit que,

$$real(rw^l) \sqsubseteq \gamma(rw^l). \tag{4.2}$$

De (4.1) et (4.2) nous concluons que $rw^l \equiv real(rw^l)$. □

Exemple 19 La figure 4.6 illustre le mapping γ de rw^5 vers $real(rw^5)$.

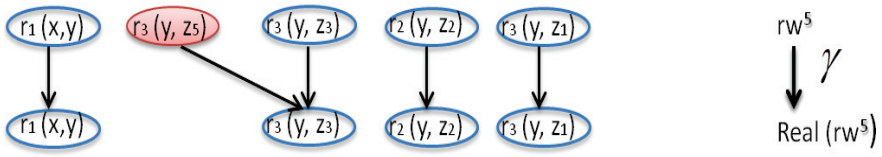

FIGURE 4.6 – Le morphisme γ de rw^5 vers $real(rw^5)$.

◊

Nous proposons donc, qu'à chaque génération de réécritures par l'algorithme $\mathcal{H}MiniCon$, de remplacer chaque réécriture rw par la réécriture réelle $real(rw)$. Ainsi, il n'est plus utile de réécrire les atomes correspondants aux noeuds virtuels.

La propriété suivante stipule que dans un arbre d'atomes, il ne peut y avoir deux noeuds réels qui ont le même historique. Autrement dit, tous les noeuds réels sont uniques par rapport à leur historique.

Propriété 4.12 (Unicité des noeuds réels)**.** *Pour chaque paire de noeuds réels* g^k_i *et* g^k_j *dans* $\mathcal{AT}$*, si* $History(g^k_i) = History(g^k_j)$ *alors* $g^k_i = g^k_j$*.*

Démonstration. Nous démontrons cette propriété par récurrence sur k. Pour chaque paire de noeuds distincts g^1_i et g^1_j dans $\mathcal{AT}$, nous avons (par définition) $History(g^1_i) \neq History(g^1_i)$. En effet, l'historique d'un noeud fils de la racine est en partie défini à partir de la position de l'atome dans la requête. Il est

donc impossible pour deux noeuds distincts d'avoir la même position et par conséquent le même historique.

Considérons g_i^k et g_j^k deux noeuds réels dans $\mathcal{AT}$ avec $History(g_i^k) = History(g_j^k)$ et $g_i^k = g_j^k$. Il faut démontrer maintenant que pour chaque g_i^{k+1} et g_j^{k+1}, deux noeuds réels dans $\mathcal{AT}$ avec $History(g_i^{k+1}) = History(g_j^{k+1})$, nous avons $g_i^{k+1} = g_j^{k+1}$.

- $type(g_i^{k+1}) = Direct$ et $type(g_j^{k+1}) = Direct$, ceci implique que $History(parent(g_i^{k+1})) = History(parent(g_j^{k+1}))$ avec $parent(g_i^{k+1})$ et $parent(g_j^{k+1})$ sont dans rw^k et ils sont des noeuds réels dans $\mathcal{AT}$. Nous avons alors $parent(g_i^{k+1}) = parent(g_j^{k+1})$. Par conséquent, $g_i^{k+1} = g_j^{k+1}$ car un noeud a au plus un fils direct.
- $type(g_i^{k+1}) = Indirect$ et $type(g_j^{k+1}) = Indirect$, ceci implique que $History(parent(g_i^{k+1})) = History(parent(g_j^{k+1}))$ avec $parent(g_i^{k+1})$ et $parent(g_j^{k+1})$ sont dans rw^k et ils sont des noeuds réels dans $\mathcal{AT}$. Nous avons alors $parent(g_i^{k+1}) = parent(g_j^{k+1})$. Par conséquent, $g_i^{k+1} = g_j^{k+1}$.
- $type(g_i^{k+1}) = Direct$ et $type(g_j^{k+1}) = Indirect$, dans ce cas, par définition, g_j^k est un noeud virtuel.

□

Démonstration du théorème 4.3 À partir de la propriété 4.12 et de la condition sur la non apparition double du même couple dans l'historique d'un noeud, nous pouvons déduire que l'ensemble des noeuds réels est fini. En effet, il est impossible d'avoir un ensemble infini de noeuds réels sachant qu'ils sont uniques et qu'il n'y a pas de doublons dans l'historique. (4.3)

Prouvons maintenant que chaque branche $\mathcal{X}$ dans $\mathcal{RT}$ est finie. Nous supposons que $\mathcal{X}$ est infinie. Pour chaque rw^i dans $\mathcal{X}$ nous savons, du théorème 4.6, que $rw^i \equiv real(rw^i)$.
De (4.3), il s'ensuit qu'il existe dans $\mathcal{X}$ deux réécritures rw^i et rw^{i+k} telles que $real(rw^i) = real(rw^{i+k})$. En effet, l'ensemble des noeuds réels étant fini, l'ensemble des réécritures possibles (combinaison des noeuds réels) est également finie. Par conséquent,

$$rw^i \equiv rw^{i+k}. \tag{4.4}$$

De la propriété de l'algorithme de réécriture de requêtes, nous avons :

$$rw^{i+j} \sqsubseteq rw^{i}, 1 \leq j \leq k. \tag{4.5}$$

De (4.4) et (4.5) nous avons : $rw^{i} \equiv rw^{i+j}, 1 \leq j \leq k$. En particulier,

$$rw^{i} \equiv rw^{i+2}. \tag{4.6}$$

De la définition de l'arbre de réécritures nous savons que rw^{i+2} est une réécriture de rw^{i} générée par l'application de la double réécriture et de (4.6) nous concluons que c'est la condition d'arrêt de l'algorithme. La branche $\mathcal{X}$ est donc finie. Nous pouvons donc conclure que l'algorithme $\mathcal{H}MiniCon^{+}$ termine dans le cas où il n'y a pas de boucles (test sur l'historique). □

4.4 Complexité théorique de l'approche

En se basant sur l'historique des noeuds, nous avons déterminé la borne supérieure de la complexité de notre approche. Pour cela, nous avons calculé la plus grande taille possible de l'historique d'un noeud comme suit :

Étant donnée une occurrence d'un atome g dans un arbre d'atomes, si l'historique $History(g)$ ne contient pas deux fois le même couple, alors la taille maximale $msize_{History(g)}$ de l'historique est :

$$msize_{History(g)} = \sum_{v \in (\mathcal{V}_1 \cup \mathcal{V}_2)} size(v) \times (size(v) - 1) \times B_n$$

où B_n est le n^{ieme} nombre de bell[2], qui est le nombre de partitions d'un ensemble de $n = |vars(g)|$ membres.

À partir de là, la borne supérieure de la complexité de notre approche est :

$$2^{Max_{g \in atoms(q)}(msize_{History(g)})!}$$

2. http ://www-history.mcs.st-and.ac.uk/Miscellaneous/StirlingBell/bell.html

Chapitre 5

Maximalité des vues

Sommaire

5.1 Introduction

La deuxième propriété souhaitée dans notre approche est la maximalité des vues générées. Dans ce chapitre, nous démontrons que l'algorithme $\mathcal{H}MiniCon^+$ est maximal si les arbres de réécritures générées sont finis. Plus précisément, nous démontrons que dans le cas d'arbres de réécritures finis, s'il existe deux requêtes $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$ définies respectivement sur $\mathcal{V}_1$ et $\mathcal{V}_2$ telles que $q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2}$, alors il existe une requête $q^{\mathcal{W}}$ définie sur $\mathcal{W}$ [1] telle que $q^{\mathcal{W}} \equiv q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2}$.

Nous présentons dans ce chapitre les définitions et les propriétés nécessaires à la preuve de la maximalité des vues générées dans le cas des arbres de réécritures finis. Pour cela, nous construisons, à partir des deux requêtes de départ $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$, un graphe et démontrons que les composantes connexes de ce graphe représentent des requêtes sur les vues générées par l'algorithme $\mathcal{H}MiniCon^+$.

1. Ensemble de vues générées.

5.2 Graphe non-orienté $\mathcal{G}_{(q^{\mathcal{V}_1}, q^{\mathcal{V}_2})}$ de requêtes équivalentes

Soient $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$ deux requêtes définies respectivement sur les vues $\mathcal{V}_1$ et $\mathcal{V}_2$ telles que $q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2}$. De la définition d'équivalence, il existe un morphisme $\mu^{\mathcal{V}_1 \to \mathcal{V}_2}$ des atomes de $expansion(q^{\mathcal{V}_1})$ vers les atomes de $expansion(q^{\mathcal{V}_2})$ et un morphisme $\mu^{\mathcal{V}_2 \to \mathcal{V}_1}$ des atomes $expansion(q^{\mathcal{V}_2})$ vers les atomes $expansion(q^{\mathcal{V}_1})$.

La propriété suivante nous permet de manipuler des morphismes d'équivalences simples.

Propriété 5.1 (Réduction de mappings)**.** *Soient $\mu^{\mathcal{V}_1 \to \mathcal{V}_2}$ et $\mu^{\mathcal{V}_2 \to \mathcal{V}_1}$ les deux morphismes d'équivalence entre $expansion(q^{\mathcal{V}_1})$ et $expansion(q^{\mathcal{V}_2})$. Il existe alors deux morphismes μ_1 et μ_2 entre $expansion(q^{\mathcal{V}_1})$ et $expansion(q^{\mathcal{V}_2})$ tels que $\mu_1 = \mu_1 \circ \mu_2 \circ \mu_1$ et $\mu_2 = \mu_2 \circ \mu_1 \circ \mu_2$.*

Démonstration. Soit G_1 le graphe orienté correspondant aux deux morphismes d'équivalence $\mu^{\mathcal{V}_1 \to \mathcal{V}_2}$ et $\mu^{\mathcal{V}_2 \to \mathcal{V}_1}$ définis comme suit :

- Les sommets sont les variables $vars(expansion(q^{\mathcal{V}_1})) \cup vars(expansion(q^{\mathcal{V}_2}))$.
- Il existe un arc entre une variable x et une variable y du graphe si $\mu^{\mathcal{V}_1 \to \mathcal{V}_2}(x) = y$ ou $\mu^{\mathcal{V}_2 \to \mathcal{V}_1}(x) = y$.

Soit x une variable appartenant à un cycle de taille $2n$ (taille du plus grand cycle élémentaire) ; c.-à-d. $(\mu^{\mathcal{V}_2 \to \mathcal{V}_1} \circ \mu^{\mathcal{V}_1 \to \mathcal{V}_2})^n(x) = x$. On pose $\mu_1(x) = \mu^{\mathcal{V}_1 \to \mathcal{V}_2} \circ (\mu^{\mathcal{V}_2 \to \mathcal{V}_1} \circ \mu^{\mathcal{V}_1 \to \mathcal{V}_2})^{n-1}(x)$ et $\mu_2(x) = \mu^{\mathcal{V}_2 \to \mathcal{V}_1}(x)$. À partir de là, nous définissons un nouveau graphe G_2 comme suit :

- les sommets sont les sommets de G_1.
- Il existe un arc entre une variable x et une variable y du graphe si $\mu_1(x) = y$ ou $\mu_2(x) = y$.

Nous avons : $\mu^{\mathcal{V}_1 \to \mathcal{V}_2} \circ (\mu^{\mathcal{V}_2 \to \mathcal{V}_1} \circ \mu^{\mathcal{V}_1 \to \mathcal{V}_2})^{n-1} \circ \mu^{\mathcal{V}_2 \to \mathcal{V}_1}(x) = x$. Donc, $\mu_1 \circ \mu_2(x) = x$ et par conséquent, pour tout x appartenant à un cycle de taille $2n$ dans G_1, x appartient à un cycle de taille 2 dans le graphe G_2.

Soit x une variable appartenant à un cycle de taille $2m \leq 2n - 2$ dans le graphe G_1, et soit $S_x = \{y \mid \mu^{\mathcal{V}_1 \to \mathcal{V}_2} \circ (\mu^{\mathcal{V}_2 \to \mathcal{V}_1} \circ \mu^{\mathcal{V}_1 \to \mathcal{V}_2})^k \circ \mu^{\mathcal{V}_2 \to \mathcal{V}_1}(x) = y; k \in \mathbb{N}\}$. Nous avons : $|S_x| = m$ et $\forall y \in S_x, S_x = S_y$. Nous savons que $\mu'(x) \in S_x$. Donc,

si x appartient à un cycle de taille $2m \leq 2n-2$ dans le graphe G_1 alors x appartient à un cycle $\leq 2m$ dans le graphe G_2.

De la même manière, on itère la construction d'un nouveau graphe G_i à partir du graphe G_{i-1} jusqu'à l'obtention d'un graphe ne contenant que des cycles de taille 2.

Soit G_n le graphe ne contenant que des cycles de taille 2. On suppose $2k$ le plus grand chemin non cyclique. Soit x et y deux sommets tels que : $(\mu^{\mathcal{V}_2\to\mathcal{V}_1} \circ \mu^{\mathcal{V}_1\to\mathcal{V}_2})^k(x) = y$ et $\mu^{\mathcal{V}_2\to\mathcal{V}_1} \circ \mu^{\mathcal{V}_1\to\mathcal{V}_2}(y) = y$. Alors : $\mu_1 = \mu^{\mathcal{V}_1\to\mathcal{V}_2} \circ (\mu^{\mathcal{V}_2\to\mathcal{V}_1} \circ \mu^{\mathcal{V}_1\to\mathcal{V}_2})^k$ et $\mu_2 = \mu^{\mathcal{V}_2\to\mathcal{V}_1} \circ (\mu^{\mathcal{V}_1\to\mathcal{V}_2} \circ \mu^{\mathcal{V}_2\to\mathcal{V}_1})^k$. □

Exemple 20 La figure 5.1 montre un exemple de la réduction des mappings.

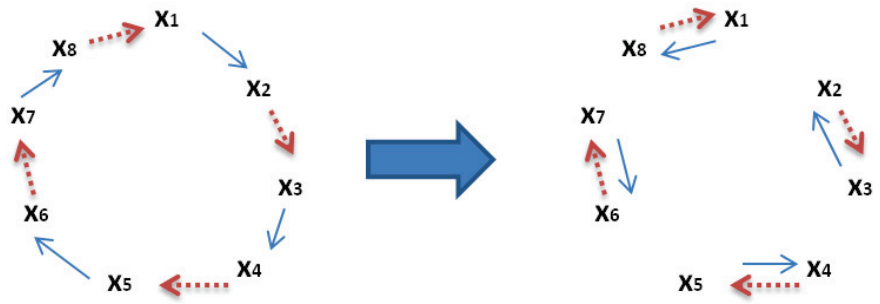

FIGURE 5.1 – Réduction des morphismes d'équivalence.

◊

Cette réduction de morphismes d'équivalence nous permet de facilement construire et parcourir le graphe $\mathcal{G}_{(q^{\mathcal{V}_1},q^{\mathcal{V}_2})}$.

Définition 5.2 (Graphe $\mathcal{G}_{(q^{\mathcal{V}_1},q^{\mathcal{V}_2})}$). *Soient deux requêtes équivalentes $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$. Le graphe non orienté $\mathcal{G}_{(q^{\mathcal{V}_1},q^{\mathcal{V}_2})}$ correspondant est construit comme suit :*

- *Les sommets sont les atomes de $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$.*
- *Il existe un arc entre un atome $g^{\mathcal{V}_1} \in q^{\mathcal{V}_1}$ et un atome $g^{\mathcal{V}_2} \in q^{\mathcal{V}_2}$ s'il existe un atome $g^{exp\mathcal{V}_1} \in expansion(q^{\mathcal{V}_1})$ et un atome $g^{exp\mathcal{V}_2} \in expansion(q^{\mathcal{V}_2})$ tels que : soit $g^{exp\mathcal{V}_1} = \mu_2(g^{exp\mathcal{V}_2})$ soit $g^{exp\mathcal{V}_2} = \mu_1(g^{exp\mathcal{V}_1})$.*

Exemple 21 Prenons l'exemple suivant pour illustrer la construction du graphe. Soient les deux ensembles de vues suivants :

$\mathcal{V}_1$:$v_{11}(x, y) \leftarrow r_1(x, y)$.
$v_{12}(y, t) \leftarrow r_2(y, z), r_3(z, t)$.
$v_{13}(t, u) \leftarrow r_4(t, u)$.
$\mathcal{V}_2$:$v_{21}(x', t') \leftarrow r_1(x', y'), r_2(y', z'), r_3(z', t')$.
$v_{22}(t', u') \leftarrow r_4(t', u')$.

Soient $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$ deux requêtes équivalentes définies comme suit :

$q^{\mathcal{V}_1}(x) \leftarrow v_{11}(x, y), v_{12}(y, t), v_{13}(t, u)$.
$q^{\mathcal{V}_2}(x') \leftarrow v_{21}(x', t'), v_{22}(t', u')$.

La figure 5.2 présente le graphe $\mathcal{G}_{(q^{\mathcal{V}_1}, q^{\mathcal{V}_2})}$ construit à partir des deux requêtes équivalentes. Les noeuds du graphe sont l'union des atomes des deux requêtes,

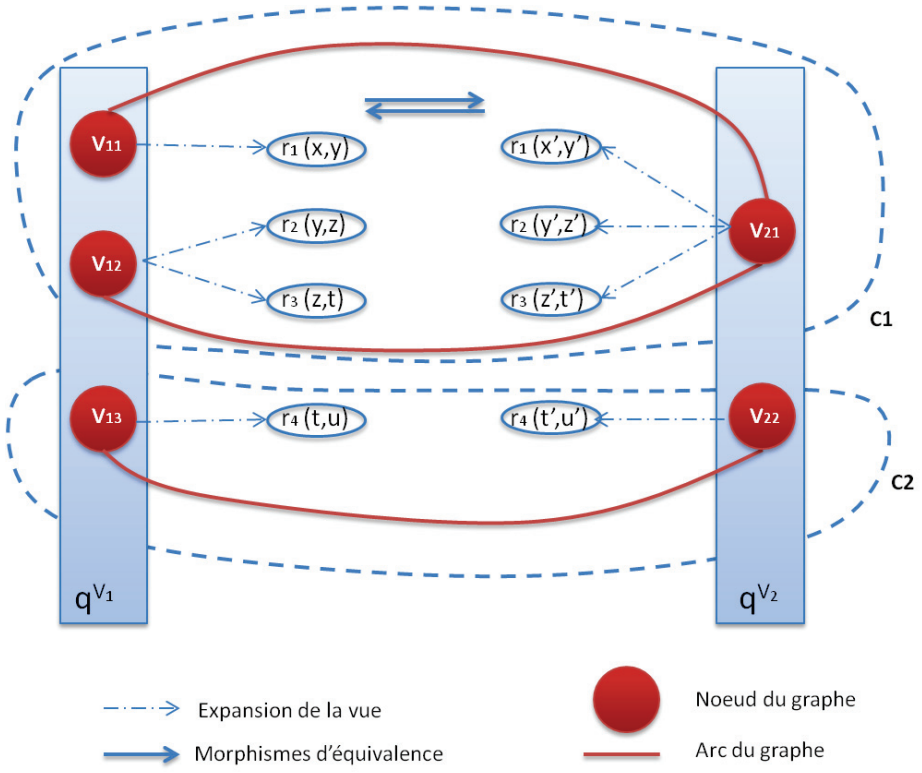

FIGURE 5.2 – Graphe $\mathcal{G}_{(q^{\mathcal{V}_1}, q^{\mathcal{V}_2})}$.

à savoir : $\{v_{11}, v_{12}, v_{13}, v_{21}, v_{22}\}$. Les arcs sont construits en fonction des morphismes d'équivalence entre l'expansion des deux requêtes. Dans cet exemple, il existe deux composantes connexes dans le graphe C1 et C2.

L'application de l'algorithme $\mathcal{H}MiniCon^+$ pour cet exemple génère l'en-

semble des vues suivant :

$\mathcal{W}$: $gv_1(x,t) \leftarrow v_{11}(x,y), v_{12}(y,t)$.
$gv_2(t,u) \leftarrow v_{13}(t,u)$.

Nous démontrons que les requêtes définies à partir des composantes connexes représentent des instanciations des vues générées par l'algorithme $\mathcal{H}MiniCon^+$. Dans cet exemple, en considérant uniquement les atomes de la requête $q^{\mathcal{V}_1}$, nous remarquons que ceux de la composante C1 représentent les atomes de la vue gv_1 et de la même manière ceux de la composante C2 représentent ceux de la vue générée gv_2. Nous considérons ici que les atomes de la requête $q^{\mathcal{V}_1}$ car les vues générées sont définies sur l'ensemble $\mathcal{V}_1$. $\Diamond$

5.3 Correspondance entre $\mathcal{RT}$ et $\mathcal{G}_{(q^{\mathcal{V}_1},q^{\mathcal{V}_2})}$

L'idée principale dans cette section est de démontrer que parcourir le graphe à partir d'un noeud $g^{\mathcal{V}_1}$ dans le but de calculer une composante connexe revient à parcourir une certaine branche de l'arbre de réécritures de la requête q, telle que le corps de q est l'atome $g^{\mathcal{V}_1}$.

5.3.1 Réécriture et application de mapping

Nous nous baserons sur le lemme suivant pour passer d'une structure à l'autre, à savoir, l'arbre de réécritures et le graphe.

Lemme 5.3. *Soient rw^i une requête à réécrire, β_i un mapping défini sur les variables de rw^i et q une requête définie comme suit : $q = \beta_i(rw^i)$. Soit rw_q une réécriture de q dans l'arbre de réécritures en utilisant l'ensemble de vues $\mathcal{V}'$. Il existe alors une réécriture de rw^i notée rw^{i+1} telle qu'il existe un mapping β_{i+1} défini sur les variables de rw^{i+1} où $rw_q = \beta_{i+1}(rw^{i+1})$.*

Démonstration. Soient rw^i une requête, β_i un mapping sur les variables de rw^i et q une requête définie comme suit : $q = \beta_i(rw^i)$.

Soit rw_q une réécriture de q. Il existe une réécriture rw' de rw^i construite comme suit : pour chaque MCD mcd^j utilisé dans la construction de rw_q pour réécrire k atomes de q, l'algorithme utilise un ensemble de MCD $mcd^j_1, ...mcd^j_n$

utilisant la même vue que mcd^j pour réécrire les k atomes de rw^i correspondants aux atomes couverts par mcd^j.
Il existe un mapping β' de $expansion(rw')$ vers $expansion(rw_q)$ défini comme suit :
$\forall x \in vars(expansion(rw'))$

- Si $x \in codomaine(\theta)$ alors $\beta'(x) = \theta'(\beta_i(\theta^{-1}(x)))$, où θ (resp. θ') correspond au mapping de réécriture de la requête rw^i (resp. q). Soient x et y deux variables distinctes de $vars(rw^i)$ tels que $\theta(x) = \theta(y) = z$. Ceci signifie que la vue unifie les deux variables. On rappelle ici que la même vue est utilisée pour réécrire les atomes dans q correspondants aux atomes dans rw^i. Par conséquent, $\theta'(\beta_i(x)) = \theta'(\beta_i(y))$. Nous posons alors $\theta^{-1}(z) = \theta'(\beta_i(x)) = \theta'(\beta_i(y))$.
- Sinon, w a été introduit comme une variable fraiche. Dans ce cas, Nous étendons le mapping β' pour prendre en considération les variables apparaissant dans rw' et qui ont été introduites comme variables fraiches singletons (noté : $\mathcal{FV}$). L'extension du mapping β' est définie comme suit : soit $y \in \mathcal{FV}$ et $y \in vars(v_k)$ où v_k correspond à la vue du MCD mcd^i_j, alors $\beta'(y) = w$ tel que $w \in vars(v_k)$ où v_k correspond à la vue du MCD mcd^i et $pos(y, v_k) = pos(w, v_k)$ avec $pos(var, a)$ une fonction qui retourne la position d'une variable var dans l'atome a.

Nous concluons que : $\beta'(rw') = rw_q$.

Nous rappelons ici que nous remplaçons dans l'arbre de réécritures la réécriture rw' par une requête équivalente notée rw^{i+1} où $rw^{i+1} = real(rw')$. De la définition d'équivalence entre requêtes, nous avons deux morphismes d'équivalence $\gamma^{rw' \to rw^{i+1}}$ et $\gamma^{rw^{i+1} \to rw'}$. Nous définissons β_{i+1} de rw^{i+1} vers rw_q comme suit : $\beta_{i+1} = \beta'(\gamma^{rw^{i+1} \to rw'})$. Nous avons alors : $\beta_{i+1}(rw^{i+1}) = rw_q$.
□

5.3.2 Caractéristique d'une composante connexe dans $\mathcal{G}_{(q^{\mathcal{V}_1}, q^{\mathcal{V}_2})}$

Nous définissons dans cette section les requêtes définies sur $\mathcal{V}_1$ et $\mathcal{V}_2$ qui correspondent aux composantes connexes et à partir de là nous démontrons que les requêtes définies sur $\mathcal{V}_1$ sont des instanciations des vues générées par notre algorithme. Nous prouvons par la suite que ces vues peuvent être com-

binées entre elles et construire une requête équivalente à $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$.

Définition 5.4 (requêtes à partir d'une composante connexe)**.** *Soit $\mathcal{G}_{(q^{\mathcal{V}_1},q^{\mathcal{V}_2})}$ un graphe non orienté. Soit C une composante connexe du graphe. Nous définissons deux requêtes $q_C^{\mathcal{V}_1}$ et $q_C^{\mathcal{V}_2}$ à partir de C comme suit : $\forall\ g^{\mathcal{V}_1} \in C,\ g^{\mathcal{V}_1} \in atomes(q_C^{\mathcal{V}_1})$ et $\forall\ g^{\mathcal{V}_2} \in C,\ g^{\mathcal{V}_2} \in atomes(q_C^{\mathcal{V}_2})$.*

Les variables de tête de $q_C^{\mathcal{V}_1}$ sont définies comme suit : $x \in headvars(q_C^{\mathcal{V}_1})$ si $x \in vars(q_C^{\mathcal{V}_1})$ et $\mu_1(x) \in vars(q_C^{\mathcal{V}_2})$ et $\mu_2(\mu_1(x)) \in vars(q_C^{\mathcal{V}_1})$.

De la même manière, les variables de tête de $q_C^{\mathcal{V}_2}$ sont définies comme suit : $x \in headvars(q_C^{\mathcal{V}_2})$ si $x \in vars(q_C^{\mathcal{V}_2})$ et $\mu_2(x) \in vars(q_C^{\mathcal{V}_1})$ et $\mu_1(\mu_2(x)) \in vars(q_C^{\mathcal{V}_2})$.

Remarque : En respectant la forme des mappings présentés dans la section 5.2 (page 58), nous avons : si $x \in headvars(q_C^{\mathcal{V}_1})$ alors $\mu_1(\mu_2(x)) \in headvars(q_C^{\mathcal{V}_1})$ et $\mu_2(x) \in headvars(q_C^{\mathcal{V}_2})$.

La propriété suivante montre que pour chaque composante connexe C du graphe correspondant à $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$, la requête $q_C^{\mathcal{V}_1}$ est une instanciation d'une vue générée par l'algorithme $\mathcal{H}MiniCon^+$.

Propriété 5.5 (Vues et composantes connexes)**.** *Soit $\mathcal{G}_{(q^{\mathcal{V}_1},q^{\mathcal{V}_2})}$ le graphe correspondant à $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$. Soit C une composante connexe du graphe. Alors il existe un mapping β_C tel que : $q_C^{\mathcal{V}_1} = \beta_C(gv_i)$ où gv_i est une vue calculée par l'algorithme $\mathcal{H}MiniCon^+$.*

Démonstration. Afin de démontrer la propriété 5.5, nous définissons un point de départ $q_1^{\mathcal{V}_1}$ et démontrons par récurrence comment calculer $q_{i+2}^{\mathcal{V}_1}$ à partir de $q_i^{\mathcal{V}_1}$ dans le but de calculer $q_C^{\mathcal{V}_1}$.

Soit $q_1^{\mathcal{V}_1}$ une requête définie comme suit :

- $atomes(q_1^{\mathcal{V}_1}) = g^{\mathcal{V}_1}$ (un noeud du graphe $\mathcal{G}_{(q^{\mathcal{V}_1},q^{\mathcal{V}_2})}$) et
- $headvars(q_1^{\mathcal{V}_1}) = vars(g^{\mathcal{V}_1})$.

On définit $\mathcal{A}_1 = \{atomes(q_1^{\mathcal{V}_1})\}$.

Il existe une réécriture rw^1 définie sur $\mathcal{V}_1$ dans l'arbre de réécritures telle que $q_1^{\mathcal{V}_1}$ est une instanciation de la requête rw_1 : $q_1^{\mathcal{V}_1} = \beta_1(rw^1)$. Nous rappelons ici que l'algorithme $\mathcal{H}MiniCon^+$ prend en entrée un ensemble de vues définies sur $\mathcal{V}_1$. Et par conséquent, rw^1 est une réécriture dans l'arbre de réécritures. En effet, rw^1 est défini comme suit :

- $atomes(rw^1) = v_{1i}$ où v_{1i} est la tête d'une vue et $g^{\mathcal{V}_1} = \beta_1(v_{1i})$
- $headvars(rw^1) = vars(v_{1i})$

Soit $q_i^{\mathcal{V}_1}$ une requête définie à l'étape i où i est un nombre impair. Réécrire $q_i^{\mathcal{V}_1}$ revient à réécrire $expansion(q_i^{\mathcal{V}_1})$. Une des réécritures possibles de $expansion(q_i^{\mathcal{V}_1})$ en utilisant les vues $\mathcal{V}_2$ est la requête $(q_{i+1}^{\mathcal{V}_2})$ définie comme suit :

- $g^{\mathcal{V}_2} \in atomes(q_{i+1}^{\mathcal{V}_2})$ s'il existe un arc entre $g^{\mathcal{V}_2}$ et un atome $g^{\mathcal{V}_1}$ de $atomes(q_i^{\mathcal{V}_1})$. En effet, par les implications des équivalences nous avons : $\forall\ g^{exp\mathcal{V}_1} \in expansion(q_i^{\mathcal{V}_1}),\ \exists\ g'^{exp\mathcal{V}_2} \in expansion(q_{i+1}^{\mathcal{V}_2})$ où $g'^{exp\mathcal{V}_2} = \mu_1(g^{exp\mathcal{V}_1})$.
- Si $x \in headvars(q_i^{\mathcal{V}_1})$ et $\mu_1(x) \in headvars(q^{\mathcal{V}_2})$ alors $\mu_1(x) \in headvars(q_{i+1}^{\mathcal{V}_2})$. En effet, de la définition de la requête $q^{\mathcal{V}_2}$, $x \in headvars(q^{\mathcal{V}_2})$ implique que la variable a été projetée par les vues et donc, $\mu_1(x) \in headvars(q^{\mathcal{V}_2})$.

Du lemme 5.3 (page 61), il s'ensuit qu'il existe une réécriture de rw^i notée rw^{i+1} telle que : $q_{i+1}^{\mathcal{V}_2} = \beta_{i+1}(rw^{i+1})$. On rappelle ici que les mêmes vues ont été utilisées pour réécrire les atomes de rw^i. Ceci implique que si $x \in headvars(rw^i)$ et $\mu_1(\beta_i(x)) \in headvars(q_{i+1}^{\mathcal{V}_2})$ alors : $\theta_{i+1}(x) \in headvars(rw^{i+1})$, où θ_{i+1} est le mapping de réécriture de rw^i vers rw^{i+1}.

On définit $\mathcal{A}_{i+1}$ comme suit : $\mathcal{A}_{i+1} = \mathcal{A}_i \cup \{atomes(q_{i+1}^{\mathcal{V}_2})\}$.

De la même manière, une des réécritures possibles de $q_{i+1}^{\mathcal{V}_2}$ en utilisant les vues $\mathcal{V}_1$ est la requête $(q_{i+2}^{\mathcal{V}_1})$ définie comme suit :

- $g^{\mathcal{V}_1} \in atomes(q_{i+2}^{\mathcal{V}_1})$ s'il existe un arc entre $g^{\mathcal{V}_1}$ et un atome $g^{\mathcal{V}_2}$ de $atomes(q_{i+1}^{\mathcal{V}_2})$.
- Si $x \in headvars(q_{i+1}^{\mathcal{V}_2})$ et $\mu_2(x) \in headvars(q^{\mathcal{V}_1})$ alors $\mu_2(x) \in headvars(q_{i+2}^{\mathcal{V}_1})$.

Du lemme 5.3, il s'ensuit qu'il existe une réécriture de rw^{i+1} notée rw^{i+2} telle que : $q_{i+2}^{\mathcal{V}_1} = \beta_{i+2}(rw^{i+2})$.

On définit $\mathcal{A}_{i+2}$ comme suit : $\mathcal{A}_{i+2} = \mathcal{A}_{i+1} \cup \{atomes(q_{i+2}^{\mathcal{V}_1})\}$.

Si $\mathcal{A}_{k+2} = \mathcal{A}_k$, alors c'est une composante connexe. Autrement dit, aucun atome n'a été rajouté lors des réécritures. Les mêmes vues ont été utilisées et par conséquent, nous avons :

$$q_{k+2}^{\mathcal{V}_1} = q_k^{\mathcal{V}_1} = q_C^{\mathcal{V}_1}$$

L'arbre de réécritures étant fini, nous pouvons conclure qu'il existe deux réécritures rw^n et rw^{n+2} dans l'arbre de réécritures où $n \geq k$ et $rw^n \equiv rw^{n+2}$

telles que :

$$q_k^{\mathcal{V}_1} = \beta_n(rw^n) \quad et \quad q_{k+2}^{\mathcal{V}_1} = \beta_{n+2}(rw^{n+2})$$

Cette équivalence est la condition d'arrêt dans l'algorithme $\mathcal{H}MiniCon^+$ et la réécriture rw^{n+2} est retournée comme une vue sur $\mathcal{V}_1$ et son instanciation correspond à la requête de $q_C^{\mathcal{V}_1}$, une composante connexe dans le graphe. Nous définissons $\beta_C = \beta_{n+2}$ et nous avons donc $\beta_C(rw^{n+2}) = q_C^{\mathcal{V}_1}$.

□

La propriété suivante exprime que si une variable apparait dans deux composantes connexes alors elle apparait dans la tête des requêtes définies à partir de ces composantes connexes. Ceci permettra de combiner par la suite les composantes connexes.

Propriété 5.6 (Variables de jointure et composantes connexes)**.** *Si* $x \in vars(q_{C_i}^{\mathcal{V}_1}) \cap vars(q_{C_j}^{\mathcal{V}_1})$ *où* $i \neq j$ *alors* $x \in headvars(q_{C_i}^{\mathcal{V}_1})$ *et* $x \in headvars(q_{C_j}^{\mathcal{V}_1})$.

Démonstration. Si $x \in vars(q_{C_i}^{\mathcal{V}_1}) \cap vars(q_{C_j}^{\mathcal{V}_1})$ où $i \neq j$ alors $\mu_1(x) = y$ et $y \in vars(q_{C_i}^{\mathcal{V}_2}) \cap vars(q_{C_j}^{\mathcal{V}_2})$. En effet, si $y \notin vars(q_{C_i}^{\mathcal{V}_2}) \cap vars(q_{C_j}^{\mathcal{V}_2})$, ceci implique que y a été introduite comme variable fraiche lors de l'expansion et donc y ne peut apparaitre que dans une seule composante connexe. Par conséquent, x aussi n'apparaitrait que dans une seule composante connexe. Ce n'est pas l'hypothèse de départ.

De la même manière, si $y \in vars(q_{C_i}^{\mathcal{V}_2}) \cap vars(q_{C_j}^{\mathcal{V}_2})$ où $i \neq j$ alors $\mu_2(y) = z$ et $z \in vars(q_{C_i}^{\mathcal{V}_1}) \cap vars(q_{C_j}^{\mathcal{V}_1})$. En effet, si $z \notin vars(q_{C_i}^{\mathcal{V}_1}) \cap vars(q_{C_j}^{\mathcal{V}_1})$, ceci implique que z a été introduite comme variable fraiche lors de l'expansion et donc z ne peut apparaitre que dans une seule composante connexe. Par conséquent, y aussi n'apparaitrait que dans une seule composante connexe. Ce n'est pas l'hypothèse de départ.

A partir de là, nous concluons que si $x \in vars(q_{C_i}^{\mathcal{V}_1}) \cap vars(q_{C_j}^{\mathcal{V}_1})$ où $i \neq j$ alors $x \in headvars(q_{C_i}^{\mathcal{V}_1}) \cap headvars(q_{C_j}^{\mathcal{V}_1})$. En effet, de la définition 5.4 (page 63) nous avons : si $x \in vars(q_C^{\mathcal{V}_1})$ et $\mu_1(x) \in vars(q_C^{\mathcal{V}_2})$ et $\mu_2(\mu_1(x)) \in vars(q_C^{\mathcal{V}_1})$ alors $x \in headvars(q_C^{\mathcal{V}_1})$. □

Nous démontrons maintenant la maximalité de notre approche dans les cas où les arbres de réécritures sont finis (pas de boucle lors du processus de réécriture).

Théorème 5.7 (Maximalité de $\mathcal{H}MiniCon^+$). *Soient $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$ deux requêtes définies respectivement sur $\mathcal{V}_1$ et $\mathcal{V}_2$ telles que $q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2}$.Il existe alors une requête $q^{\mathcal{W}}$ définie sur $\mathcal{W}$ telle que $q^{\mathcal{W}} \equiv q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2}$.*

Démonstration. Soient $q^{\mathcal{V}_1}$ et $q^{\mathcal{V}_2}$ deux requêtes définies respectivement sur $\mathcal{V}_1$ et $\mathcal{V}_2$ telles que $q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2}$.
De la définition 5.4, nous avons : $q^{\mathcal{V}_1} = q^{\mathcal{V}_1}_{C_1}, q^{\mathcal{V}_1}_{C_2}, ..., q^{\mathcal{V}_1}_{C_n}$
Soit $q^{\mathcal{W}}$ définie comme suit : $q^{\mathcal{W}} = \beta_{C_1}(gv_1), \beta_{C_2}(gv_2), ..., \beta_{C_n}(gv_n)$
où pour $1 \leq i \leq n$, gv_i est une vue générée par l'algorithme $\mathcal{H}MiniCon^+$ et $q^{\mathcal{V}_1}_{C_i} = expansion(\beta_{C_i}(gv_i))$. Il suffit de démontrer que $q^{\mathcal{W}} \equiv q^{\mathcal{V}_1}$. Il faut pour cela démontrer qu'il existe :

- un morphisme ω_1 de $vars(q^{\mathcal{V}_1})$ vers $vars(expansion(q^{\mathcal{W}}))$ et
- un morphisme ω_2 de $vars(expansion(q^{\mathcal{W}}))$ vers $vars(q^{\mathcal{V}_1})$.

À partir de $q^{\mathcal{V}_1}_{C_i} = expansion(\beta_i(gv_i))$, il s'ensuit qu'il existe :

- un morphisme ω_1^i de $vars(q^{\mathcal{V}_1}_{C_i})$ vers $vars(expansion(\beta_i(v_i)))$ et
- un morphisme ω_2^i de $vars(expansion(\beta_i(gv_i)))$ vers $vars(q^{\mathcal{V}_1}_{C_i})$.

Les morphismes ω_1^i et ω_2^i sont l'identité sur les variables de tête.

- **Construction du morphisme** ω_1 : Soit $x \in vars(q^{\mathcal{V}_1})$. De la propriété 5.6, il s'ensuit que si $x \in vars(q^{\mathcal{V}_1}_{C_i}) \cap vars(q^{\mathcal{V}_1}_{C_j})$ alors $x \in headvars(q^{\mathcal{V}_1}_{C_i}) \cap headvars(q^{\mathcal{V}_1}_{C_j})$. Par conséquent, nous définissons $\omega_1(x) = x$. Sinon, x n'apparait que dans $vars(q^{\mathcal{V}_1}_{C_i})$, nous définissons dans ce cas $\omega_1(x) = \omega_1^i(x)$.
  Soit $g \in atomes(q^{\mathcal{V}_1})$ où $g \in atomes(q^{\mathcal{V}_1}_{C_i})$, il existe alors un atome g' appartenant à $atomes(expansion(q^{\mathcal{W}}))$ tel que $\omega_1(g) = \omega_1^i(g) = g'$. En effet, soit $x \in vars(g)$, si $x \in headvars(q^{\mathcal{V}_1}_{C_i})$ alors $\omega_1(x) = \omega_1^i(x) = x$. Sinon, x n'apparait que dans $vars(q^{\mathcal{V}_1}_{C_i})$, et dans ce cas $\omega_1(x) = \omega_1^i(x)$.
- **Construction du morphisme** ω_2 : Soit $x \in vars(expansion(q^{\mathcal{W}}))$. Si $x \in vars(q^{\mathcal{W}})$ alors $\omega_2(x) = x$. Sinon, ceci implique que x a été introduite comme variable fraiche lors de l'expansion d'une vue gv_i et par conséquent nous avons $\omega_2(x) = \omega_2^i(x)$.
  Soit $g' \in atomes(expansion(q^{\mathcal{W}}))$ où $g' \in atomes(expansion(\beta_i(gv_i)))$, il existe alors un atome g appartenant à $atomes(q_1^{\mathcal{V}})$ tel que $\omega_2(g') = \omega_2^i(g') = g$. En effet, soit $x \in vars(g)$, si $x \in headvars(\beta_i(gv_i))$ alors $\omega_2(x) = \omega_2^i(x) = x$. Sinon, x apparait dans l'expansion de $\beta_i(gv_i)$ et donc $\omega_2(x) = \omega_2^i(x)$.

Nous concluons à partir de là que $q^{\mathcal{V}_1} \equiv q^{\mathcal{W}}$.

Et par conséquent, nous avons : $q^{\mathcal{V}_1} \equiv q^{\mathcal{V}_2} \equiv q^{\mathcal{W}}$. Nous concluons que notre algorithme est maximal dans le cas des arbres de réécritures finis. □

Deuxième partie

Application : Confidentialité des données au niveau des vues matérialisées

Chapitre 6

Sécurité dans les bases de données

Sommaire

6.1 Introduction

Les informations stockées dans les bases de données sont considérées comme une ressource précieuse et importante pour les organisations. Une interruption

du service ou une fuite d'information peut provoquer des conséquences allant du désagrément à la catastrophe. Les données peuvent être des données financières, des secrets commerciaux, ou peuvent décrire des informations privées qui concernent des personnes. Le concept de sécurité des données est très large et comporte plusieurs éléments relevant de questions morales et éthiques imposées par la société, de questions juridiques ou de questions plus techniques comme la façon de protéger les données contre la perte ou l'accès non autorisé, la destruction ou la modification erronée.

Les exigences en matière de sécurité d'un système sont spécifiées à travers une politique de sécurité qui est ensuite mise en oeuvre par différents mécanismes. Pour les bases de données, les exigences relatives à la sécurité peuvent être classées dans les catégories suivantes :

- **Identification, Authentification** : Avant d'accéder à une base de données, chaque utilisateur doit s'identifier au système. L'authentification est le moyen de vérifier l'identité d'un utilisateur lors de sa connexion. La méthode d'authentification la plus courante est l'utilisation de mots de passe, mais des techniques plus avancées telles que les lecteurs de badges, des techniques de reconnaissance biométrique, ou des dispositifs d'analyse de signatures ont également vu le jour.
- **Autorisation, contrôles d'accès** : L'autorisation est la spécification d'un ensemble de règles qui définissent qui a le type d'accès à quelle information. Les politiques d'autorisations par conséquent régissent la divulgation et la modification des informations. Les contrôles d'accès sont des procédures qui sont conçues pour contrôler les autorisations. Ils sont responsables de limiter l'accès aux données stockées qu'aux seuls utilisateurs autorisés.
- **Intégrité, cohérence** : Une politique d'intégrité indique un ensemble de règles (contraintes d'intégrité) qui définissent les états corrects de la base de données pendant son fonctionnement et peut donc empêcher toute modification malveillante ou accidentelle d'informations en gardant la base de données dans un état cohérent.

La figure 6.1 montre les mécanismes utilisés, en général, pour assurer la sécurité d'une base de données. Tout utilisateur doit être authentifié et toute application sécurisée. Les communications échangées et les données doivent

être chiffrées pour éviter leur utilisation par des personnes non autorisées. Enfin, tout accès à la base de données doit être contrôlé.

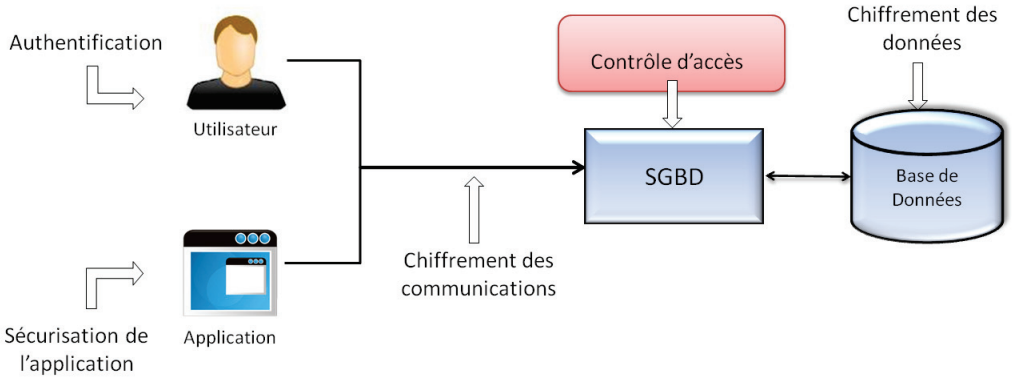

FIGURE 6.1 – Mécanismes de sécurité d'une base de données.

Dans notre travail, nous nous intéressons plus particulièrement aux aspects relevant de l'autorisation et du contrôle d'accès. Nous présentons dans ce qui suit une définition générale des politiques de sécurité pour nous intéresser plus particulièrement aux politiques de contrôle d'accès et leur utilisation.

6.2 Politique de sécurité

Une *politique de sécurité* est définie comme étant "*L'ensemble des lois, règles et pratiques qui régissent la façon dont l'information sensible et les autres ressources sont gérées, protégées et distribuées à l'intérieur d'un système spécifique*" [EC91]. On peut classer les politiques de sécurité en trois couches :

- Les politiques de sécurité physiques qui définissent un ensemble de procédures et de moyens qui protègent les lieux et les biens contre des risques de type physique (ex., incendie, inondation,...) et contrôle les accès physiques aux matériels (ex., barrières, coffres,...).
- Les politiques de sécurité administratives qui définissent l'ensemble des procédures et moyens qui gèrent l'organisation de la sécurité au sein de l'entreprise (ex., propriétés de sécurité à assurer, structure de l'organigramme, répartition des responsabilités,...).
- Les politiques de sécurité logiques qui définissent les actions autorisées qu'un utilisateur peut effectuer. Tout utilisateur devra s'authentifier.

Une fois l'identité de l'utilisateur établie, les actions légitimes que peut accomplir cet utilisateur sont déterminées par la politique d'autorisation. Notre travail de thèse concerne plus particulièrement la préservation de la confidentialité des données et dans une certaine mesure à leur intégrité. Ces deux propriétés de sécurité sont assurées par des politiques de sécurité logiques : les politiques de contrôle d'accès.

6.3 Politique de contrôle d'accès

Une *politique de contrôle d'accès* définit les droits et les interdictions d'accès. On peut dire que chaque organisation possède sa propre politique de contrôle d'accès. Certaines politiques d'accès seront plutôt axées sur la confidentialité des données (ex., le secteur militaire), quand d'autres privilégieront l'intégrité des données (ex., le secteur commercial). Contrôler l'accès c'est vérifier à chaque fois si un *sujet* (entité qui demande l'accès, dite entité active) a le droit d'effectuer une *action* sur un *objet* (entité à laquelle le sujet souhaite accéder, dite entité passive). Le mécanisme de contrôle d'accès vérifie les droits de ce sujet à travers un ensemble d'autorisations, établi en adéquation avec la politique de contrôle d'accès de l'organisation [BS05]. La figure 6.2 schématise le mode de fonctionnement.

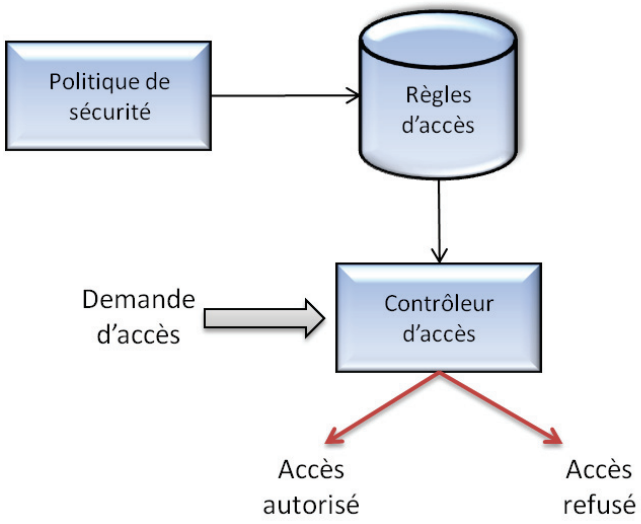

FIGURE 6.2 – Contrôleur d'accès.

Plusieurs modèles ont été définis et mis en place pour contrôler l'accès

aux informations [GD72]. Selon le modèle des données, une autorisation dans une base de données relationnelle est exprimée comme le droit d'exécuter une action (ex., sélection, mise à jour) sur une table relationnelle ou une vue [OGM08]. Une autorisation sur un document XML peut être modélisée par des vues qui expriment quelle partie du document est accessible [FCG04, GSC+09]. On peut aussi classer les modèles selon le mode d'administration des autorisations : les modèles discrétionnaires (DAC, Discretionary Access Control) [Lam74], les modèles basés sur les rôles (RBAC, Role-Based Access Control) [FSG+01] ou bien les modèles obligatoires (MAC, Mandatory Access Control) [San93].

Nous nous sommes intéressés dans notre travail à la confidentialité et l'intégrité des données au niveau des bases de données relationnelles. Nous allons présenter dans ce qui suit différents modèles de contrôle d'accès définis pour les SGBDs relationnels et s'arrêter plus particulièrement sur les *vues d'autorisations* qui sont utilisées comme modèle dans notre approche.

6.4 Modèles de contrôle d'accès pour les SGBD relationnels

6.4.1 Modèle de contrôle d'accès discrétionnaire

Le modèle de contrôle d'accès discrétionnaire (DAC) se base sur le concept de matrice de contrôle d'accès introduite par Lampson [Lam74]. Dans la forme la plus simple de la matrice, les lignes représentent un ensemble de sujets $\mathcal{S}$, les colonnes représentent des objets $\mathcal{O}$ et l'intersection d'une ligne et d'une colonne spécifie les privilèges d'accès $\mathcal{T}$ définissant quel type d'accès un sujet a sur un certain objet. Ce modèle a été progressivement amélioré pour donner naissance à d'autres modèles tels que le modèle HRU [HRU76] où il est possible de préciser les commandes qui peuvent lui être appliquées, le modèle Take-Grant [JLS76] qui se base sur la notion de graphe pour représenter la matrice de contrôle d'accès.

Appliqué aux bases de données relationnelles : $\mathcal{O}$ définit un ensemble fini de valeurs $\{o_1...o_n\}$ représentant les schémas de relations, $\mathcal{S}$ définit l'ensemble des sujets potentiels $\{s_1...s_m\}$ représentant les utilisateurs ou les applications

souhaitant accéder aux données. $\mathcal{T}$ définit l'ensemble des opérations effectuées sur une base de données : sélection, insertion, suppression, mise à jour, octroi/retrait d'autorisations... Dans le but de représenter des contrôles d'accès basés sur le contenu, un ensemble de prédicats $\mathcal{P}$ est défini pour déterminer les fenêtres d'accès qu'un sujet s peut avoir sur un objet o. Le quadruplet $\langle o, s, t, p\rangle$ est appelé *règle d'accès* et une fonction f est définie pour déterminer si une autorisation $f(o, s, t, p)$ est valide ou non :

$$f : \mathcal{O} \times \mathcal{S} \times \mathcal{T} \times \mathcal{P} \longrightarrow \{Vrai, Faux\}$$

Pour chaque $\langle o, s, t, p\rangle$, si $f(o, s, t, p)$ est évaluée à vrai, le sujet s a l'autorisation t d'accéder à l'objet o avec la granularité définie par le prédicat p [Per94].

Les mécanismes de contrôle d'accès des SGBDs courants, fondés sur le modèle DAC, gèrent les accès aux données en se basant sur l'identité du sujet et des règles d'autorisations. Ce modèle a été rapidement adopté et utilisé par les SGBDs commerciaux grâce à sa flexibilité. La notion la plus importante dans ce modèle est l'administration des autorisations, c.-à-d. la possibilité d'octroyer ou de retirer des autorisations. Avec SQL, ces actions sont réalisées respectivement par les commandes Grant et Revoke [1]. Plusieurs possibilités sont mises à disposition : l'*administration centralisée* où seulement un certain nombre de privilégiés peuvent avoir ce pouvoir (ex., les administrateurs), l'*administration du propriétaire* où le propriétaire de l'objet spécifie lui-même les autorisations, la *délégation d'administration* où le propriétaire de l'objet peut donner à d'autres utilisateurs la possibilité de spécifier les autorisations, l'*administration conjointe* où plusieurs sujets peuvent être responsables de la spécification des autorisations [BF97]. Nous présentons dans la suite quelques modèles de contrôle d'accès aux bases de données fondés sur le modèle DAC.

6.4.1.1 Modèle d'autorisation du système R

Le modèle d'autorisation du système R a été l'un des premiers modèles d'autorisations des SGBD relationnels proposé dans [GW76, ABC+76]. L'administration des autorisations dans le modèle du système R est basée sur

1. http ://www.techonthenet.com/oracle/grant_revoke.php

l'administration propriétaire et la délégation d'administration. Chaque utilisateur qui crée une table, en devient le propriétaire et par conséquent, a tous les droits d'accès sur cette table. Il a par la même occasion la possibilité de déléguer l'administration des autorisations de la table à d'autres utilisateurs. Ce modèle a été le premier à introduire l'utilisation des vues dans le but d'exprimer des règles de contrôle d'accès donnant ainsi lieu à une nouvelle perspective de la sécurité. Beaucoup d'autres modèles ont été développés en se basant sur le système R : les autorisations négatives, les autorisations basées sur le rôle ou la tâche [SCFY96, TS97], les autorisations temporelles [BBFS98], les autorisations sensibles au contexte [Ora02],... Parmi ces modèles, le modèle de contrôle d'accès à grains fins est l'un des modèles les plus utilisés par les SGBDs.

6.4.1.2 Modèle de contrôle d'accès à grains fins

C'est grâce aux besoins de certaines institutions que les contrôles d'accès à grains fins et basés sur le contenu ont vu le jour [RMSR04] :

- Les institutions médicales qui gèrent les informations des patients où la maladie des patients ne doit être divulguée qu'aux personnels traitant le patient.
- Les institutions financières qui gèrent les comptes des personnes où les clients doivent avoir la possibilité d'accéder à leurs comptes mais en aucun cas aux comptes des autres clients. D'un autre côté, le comptable, quant à lui, peut accéder à tous les comptes clients pour pouvoir mener à bien sa mission.
- Les institutions académiques qui gèrent les notes des étudiants, souhaitent donner l'accès aux étudiants mais qu'à leurs propres notes tout en donnant à chaque enseignant l'accès aux notes de tous les étudiants qui suivent son cours.

Sa force réside dans la capacité de définir des règles d'accès au niveau de la cellule d'une table (relation), ce qui rend la gestion des accès plus souple contrairement au cas où ils sont définis par l'accès à la table entière ou pas. La seconde caractéristique de ce modèle est sa capacité de définir des autorisations d'accès aux données en se basant sur le contenu de celle-ci. Les langages déclaratifs comme le langage de requêtes SQL, ont facilité la mise en place de

cette spécificité en utilisant le concept de vue.

Un tel modèle présente plusieurs avantages : la possibilité d'exprimer des accès basés sur le contenu dans un langage de haut niveau, la modification des données dans une base de données ne nécessite plus de modification au niveau des règles de contrôle d'accès. En effet, si par exemple une nouvelle donnée est insérée dans la base de données, si elle satisfait la règle de contrôle accès, elle sera automatiquement accessible. Plusieurs approches ont été proposées pour implémenter les contrôles d'accès à grains fins, les plus connues sont : la réécriture de requête dans INGRES [SW74] et la base de données privée virtuelle [Ora02].

Dans notre approche, nous nous basons sur le modèle de contrôle d'accès à grains fins à travers l'utilisation des vues. Nous détaillerons dans la section 6.6 (page 81) les approches utilisées pour implémenter les contrôles d'accès à grains fins.

6.4.2 Modèle de contrôle d'accès obligatoire

Les modèles de contrôle d'accès obligatoires gèrent les accès aux données en se basant sur une classification prédéfinie des sujets et des objets dans le système [BLP76]. Contrairement au modèle discrétionnaire, les utilisateurs ne peuvent pas définir des autorisations car ils ne sont pas propriétaires des objets. Tous les objets sont la propriété de l'organisation.

Le modèle se base sur la classification d'un ensemble de *classes d'accès* appelé aussi *niveau de sécurité* qui sont associées à chaque sujet et à chaque objet. Un sujet a accès à un objet si et seulement si un ordre (de relation), dépendant du mode d'accès, est satisfait par les classes d'accès des sujets et des objets. Le modèle se base sur les deux principes suivants : un sujet doit lire les informations provenant des objets d'un niveau de sécurité plus faible, et écrire des objets qui sont d'un niveau de sécurité plus élevé.

L'application du modèle MAC aux bases de données relationnelles a introduit une extension du modèle relationnel appelé le modèle relationnel multi-niveaux [Per94]. Le principe de ce modèle se base sur le fait que des tuples peuvent avoir différents niveaux de sécurité. La relation est alors partitionnée en différents niveau de sécurité. Ce modèle reste compliqué à mettre en place si dans un même tuple les attributs ont des niveaux de sécurité différents.

La gestion de tuples/attributs apparaissant dans plusieurs partitions génère des problèmes au niveau des mises à jours. L'implémentation de ces extensions n'est supportée par aucun système car pallier à ce genre de problème nécessiterait de revisiter les notions de base du modèle relationnel.

6.4.3 Modèle de contrôle d'accès basé sur les rôles

Le modèle de contrôle d'accès basé sur les rôles (RBAC) a été introduit et mis en place dans le but de simplifier l'administration des autorisations [SCFY96]. Ce modèle se base sur la notion de *rôle*. Un rôle peut être vu comme une fonction au sein d'une organisation et peut être défini comme un ensemble d'actions ou de responsabilités associées à cette fonction. Contrairement à la notion de groupe qui est essentiellement une collection d'utilisateurs, le rôle est à la fois une collection d'utilisateurs et de permissions. Tous les privilèges sont attribués par l'intermédiaire des rôles. Un utilisateur actif au sein du système est représenté par une *session*, elle est attribuée à un et à un seul utilisateur. Ce dernier a le droit d'endosser tout ou une partie des rôles qui lui sont attribués [FK09].

L'administration des autorisations en est considérablement simplifiée. Si un utilisateur change de fonction au sein de l'organisation, il suffit simplement de lui ôter la permission de jouer le rôle associé à cette fonction. De plus, le modèle RBAC, conformément aux besoins des organisations, représente les rôles en forme de hiérarchie permettant l'héritage des autorisations et les contraintes de séparation des tâches [Kuh97] qui empêchent de recevoir trop d'autorisations.

Ce modèle a connu plusieurs extensions : L'introduction de contraintes temporelles TRBAC (Temporal Role-Based Access Control) [BBF00], l'introduction de contraintes géographiques GEO-RBAC (Geographical Role-Based Access control) [DBCP07], l'introduction de la notion de tâche TMAC (Task-based Access Control) [TS97], l'introduction de la notion d'organisation ORBAC (Organization-Based Access Control) [CM04],...

6.5 Utilisation des graphes pour les autorisations d'accès

Nous trouvons dans la littérature d'autres approches autre que l'utilisation des vues pour modéliser les contrôles d'accès. Une des plus connues est proposée dans [dVFJ+08]. Les auteurs proposent une solution modélisant et exécutant efficacement les contrôles d'accès en s'appuyant sur la coloration, la composition et le parcours de graphes. Une permission p est une règle de la forme $[A,\ R] \rightarrow s$ qui stipule qu'un sujet s peut voir accès aux tuples sur l'ensemble des attributs A appartenant à la jointure entre les relations R.

Les auteurs définissent aussi la notion de profil de requêtes. Pour chaque requête conjonctive q, une paire $[A_q, R_q]$ est définie, appelée *profil de requête*, où A_q est l'ensemble des attributs apparaissant dans la clause Select et/ou dans la clause Where et R_q est l'ensemble des relations apparaissant dans la clause From. Dans ce modèle, les permissions et les profils de requêtes sont représentés par des graphes où les noeuds sont les attributs et les arcs représentent la relation entre les attributs d'une relation (dépendances fonctionnelles), et entre les relations (dépendances d'inclusion). Les auteurs proposent des algorithmes de coloration de graphe. Ceci permet de déterminer dans le cas des permissions, le sous ensemble des attributs accessibles par l'utilisateur, et dans le cas des requêtes, le sous-ensemble des attributs dans la clause 'Select' de la requête.

Exemple 22 Prenons un exemple simple pour illustrer la construction du graphe. Soit la table $patients(id, nom, maladie)$ d'une base de données médicale. La permission $p_1 : [(nom), patients] \rightarrow Alice$ spécifie que l'utilisatrice Alice a le droit d'accéder aux noms des patients mais pas à leur identifiant(id) ni à leur maladie. La requête suivante permet de récupérer tous les attributs de la table patients. $Select * From\ patients$;

La figure 6.3 schématise les deux graphes correspondants à la permission et à la requête. Seul l'attribut *nom* du premier graphe est colorié, ceci correspond à l'autorisation d'accéder uniquement aux noms des patients. Dans le graphe correspondant à la requête, tous les attributs sont coloriés, ceci correspond aux attributs présents dans la clause Select de la requête.

◊

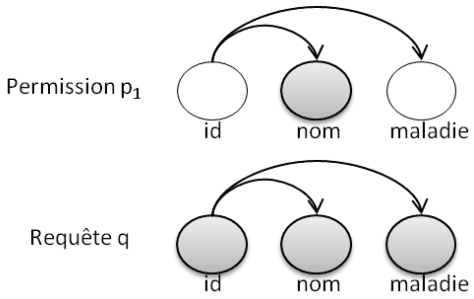

FIGURE 6.3 – Modélisation des requêtes et des permissions par des graphes.

À partir de cette modélisation, il devient plus facile de déterminer si une requête est autorisée à être exécutée par rapport à l'ensemble des permissions. Intuitivement, un système n'autorise l'exécution d'une requête que si les informations obtenues (directement ou indirectement) par la requête est un sous ensemble des informations autorisées par les permissions. Par ailleurs, les auteurs proposent des algorithmes qui permettent de composer les permissions entre elles pour pouvoir répondre aux requêtes complexes.

L'inconvénient majeur de cette approche est la non possibilité de définir des permissions sur un sous-ensemble de tuples (sélection). En effet, le modèle permet juste de déterminer l'accès ou non à un sous ensemble d'attributs. Par conséquent, il n'y a pas moyen de modéliser les contrôles d'accès basés sur le contenu.

6.6 Utilisation des vues pour les autorisations d'accès

Dans notre approche nous avons choisi l'une des techniques les plus utilisées pour implémenter le modèle de contrôle d'accès à grains fins, à savoir, les vues. Une *vue* est définie comme étant une fenêtre dynamique capable de sélectionner des sous-ensembles d'attributs et de tuples, ces derniers sont définis par une requête, appelée *requête de définition de vue* qui est associée au nom de la vue [BS05]. Les vues sont utilisées pour deux choses : (1) pour simplifier l'écriture des requêtes. Ces vues sont appelées *vues raccourcies*. À chaque fois que la vue est référencée dans une requête, cette dernière est modifiée en rem-

plaçant la vue référencée par sa définition, (2) pour implémenter les contrôles d'accès à grains fins et basés sur le contenu. Ces vues sont appelées : *vues d'autorisations*. Comme leur nom l'indique, les vues d'autorisations spécifient les données autorisées à être accédées par les politiques de contrôle d'accès. Par exemple, pour chaque docteur, une vue est définie contenant les informations de ses patients, chaque étudiant aura accès a sa propre vue contenant ses notes,...

Dans une approche naïve, l'administrateur crée, pour chaque utilisateur, un ensemble de vues qui déterminent quelles informations lui sont accessibles. Mais ceci devient vite impossible à mettre en place avec une base de données qui interagit avec des milliers d'utilisateurs et où l'administrateur doit définir le même type de vues pour chaque utilisateur. Ajouté à cela, une modification d'une simple politique de sécurité entrainerait un travail conséquent pour l'administrateur qui devra répercuter le changement sur l'ensemble des vues concernées. Dans le but de rendre l'utilisation des vues plus simple, *Rizvi et al.* [RMSR04] ont proposé l'utilisation des *vues paramétrées*. Une vue paramétrée est une vue qui est définie par une requête utilisant des paramètres (ex., l'identifiant de l'utilisateur, le temps, la localisation...). Prenons comme exemple une base de données médicale. L'administrateur de sécurité peut définir la vue paramétrée suivante pour exprimer la règle de contrôle d'accès qui spécifie que les docteurs n'ont le droit d'accéder qu'aux informations de leurs patients :

Create authorization view *MyPatients* as
select * from *Patients* where *id_doctor* = *$id_doctor* ;

L'utilisateur (ici, le docteur) pourra accéder à toutes les informations de ses patients, ceci est spécifié par la condition de sélection utilisant comme paramètre l'identifiant de l'utilisateur. L'utilisation des vues paramétrées offre une efficacité d'implémentation. En effet, l'administrateur n'aura plus besoin de définir le même type de vue plusieurs fois, mais il n'aura qu'à définir une vue paramétrée qui se personnalisera aux utilisateurs avec les bons paramètres.

Plusieurs approches ont été proposées pour utiliser les vues d'autorisations : (1) Autoriser les utilisateurs à requêter directement les tables de base ; (2) Autoriser les utilisateurs à accéder seulement à leur ensemble de vues d'autorisations et de les requêter. Nous allons présenter dans ce qui suit le principe

de chacune des deux approches ainsi que les avantages et les inconvénients de leur mise en place.

6.6.1 Requêter les vues d'autorisations

L'approche la plus naïve d'utilisation des vues d'autorisations pour assurer la confidentialité et l'intégrité des données est de définir pour chaque utilisateur un ensemble de vues d'autorisations qui spécifie à quel sous ensemble de données l'utilisateur a le droit d'accéder et par la suite, l'autoriser uniquement à requêter cet ensemble de vues d'autorisations. Autrement dit, l'utilisateur n'a aucun moyen de requêter les tables de base et par conséquent, aucun moyen d'accéder aux données non autorisées. Même si cette approche répond exactement aux besoins de confidentialité des données, sa mise en place peut s'avérer compliquée. En effet, il faudrait implémenter une interface propre à chaque utilisateur lui donnant accès seulement à ses propres vues d'autorisations. Dans une organisation ayant des milliers d'employés (une centaine de rôles) et une centaine de règles d'accès, le système deviendra ingérable pour l'administrateur de sécurité. Si un changement dans la politique de sécurité survient, il devra le répercuter au niveau de chaque interface.

6.6.2 Requêter les tables de base

L'approche alternative proposée pour répondre au besoin de simplification est de laisser aux utilisateurs la possibilité de requêter les tables de base. Les résultats de leurs requêtes sont ensuite filtrés en adéquation avec ce qu'ils avaient ou non le droit d'accéder. Plusieurs modèles ont été proposés et commercialisés. Nous en citerons les plus connus.

6.6.2.1 Les bases de données privées virtuelles

Les bases de données privées virtuelles VPD (Virtual Private Database) [Ora02] sont utilisées par Oracle pour implémenter les contrôles d'accès à grains fins en se basant sur le principe de modification de requêtes. Les autorisations sont implémentées sous forme de fonctions PL/SQL [2] qui retournent des conditions de sélection à inclure dans la clause *"where"* avec la conjonction

2. http ://www.techonthenet.com/oracle/index.php

"and" dans la requête de l'utilisateur pour assurer que ce dernier n'ait accès qu'aux données autorisées. La figure 6.4 présente un exemple de l'exécution d'une requête. Prenons le même exemple pour illustrer le processus ; comme vu plus haut, un docteur n'a le droit d'accéder qu'aux informations de ses propres patients. Cette règle est définie par la vue d'autorisation *MyPatients* (définie plus haut) qui sera encodée dans une fonction PL/SQL pour retourner les conditions appropriées. La requête est alors modifiée et exécutée. Le

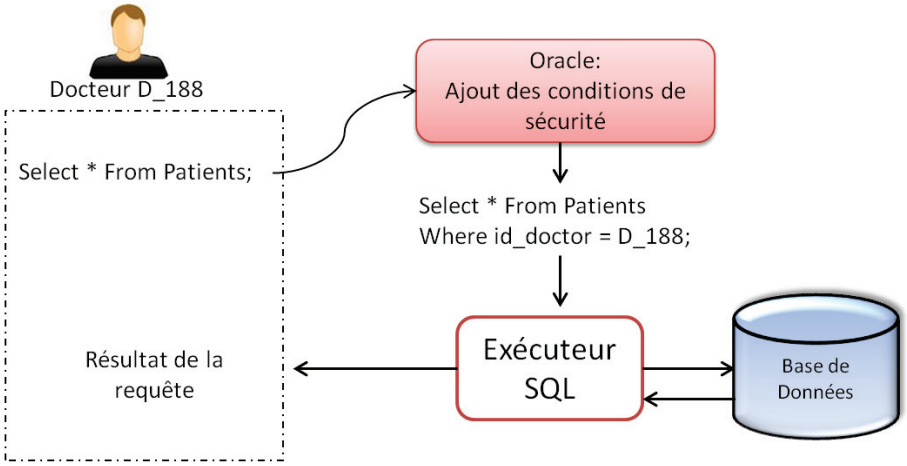

FIGURE 6.4 – Modification dynamique de requête par VPD.

docteur n'aura accès qu'aux informations de ses patients.

6.6.2.2 Modèle de Truman

Rizvi et al. [RMSR04] ont présenté le modèle Truman qui englobe l'approche de modification de requête en utilisant les vues d'autorisations. L'idée est de définir pour chaque table de la base de données une vue d'autorisation. Cette dernière spécifie tout ce à quoi l'utilisateur à le droit d'accéder à partir de cette table. L'utilisateur peut alors requêter les tables de base et les vues d'autorisations. Le système modifie la requête en remplaçant chaque table de base par sa vue d'autorisation correspondante, c'est la requête modifiée qui est exécutée. Les vues d'autorisations offrent un cadre plus efficace pour implémenter les contrôles d'accès à grains fins que les bases de données privées virtuelles. Elles offrent la possibilité de définir des règles de contrôle d'accès au niveau de la cellule en n'autorisant l'accès qu'à un sous ensemble d'attributs.

Ceci ne peut pas être mis en place avec les VPD sachant que seules les conditions de sélection peuvent être ajoutées. Néanmoins, cette approche connait certaines limites. En effet, la modification des requêtes peut introduire des inconsistances entre le résultat des requêtes et celui auquel s'attend l'utilisateur. De plus, en modifiant la requête on modifie aussi son coût d'exécution. Les requêtes modifiées peuvent être plus coûteuses en raison de la complexité des vues d'autorisations.

6.6.2.3 Modèle de Non-Truman

Le modèle de Non-Truman a été le fruit de recherche de Rizvi et al. [RMSR04] pour pallier aux limites de la modification des requêtes transparente à l'utilisateur (modèle des bases de données privées virtuelles et modèle de Truman). Le modèle de Non-Truman se base sur le concept de vérification de la validité d'une requête (sur les tables de base) sans la modifier. Une requête est dite valide si on peut répondre à la requête en utilisant seulement les données contenues dans les vues d'autorisations de l'utilisateur. Si la requête est considérée comme valide elle sera exécutée sans aucune modification. Si au contraire, le test de validité échoue, le système notifie l'utilisateur du rejet de la requête. Des règles d'inférence ont été définies comme outil de validation. Malheureusement, des problèmes de décidabilité rendent ce modèle inapplicable en pratique.

6.7 Discussion

Notre choix de considérer les contrôles d'accès à grains fins est motivé par la nature du problème que nous souhaitons traiter. Notre objectif est d'assurer la confidentialité au niveau des vues matérialisées. Ces dernières sont le résultat de requêtes, une requête peut projeter des attributs ou/et sélectionner des tuples. Par conséquent, une sécurité à grains fins doit être appliquée.

Nous avons présenté dans ce chapitre les mécanismes d'application des vues d'autorisations les plus utilisés dans la pratique et les plus étudiés dans la littérature. Dans notre travail, nous sommes restés indépendants de la manière dont les vues d'autorisations sont utilisées pour répondre aux requêtes des utilisateurs. En effet, notre objectif est d'inférer un ensemble de vues

d'autorisations pour assurer la confidentialité des données au niveau des vues matérialisées et non pas de répondre aux requêtes. Ainsi, tout modèle pourra être appliqué. Les administrateurs peuvent autoriser les utilisateurs à :

- Requêter les vues matérialisées, puis la requête est modifiée en utilisant le nouvel ensemble de vues d'autorisations.
- Requêter les vues matérialisées, puis vérifier la validité de la requête et l'exécuter sans la modifier si elle est valide.
- Requêter directement les vues d'autorisations.

Notre approche s'applique aussi dans les organisations qui utilisent le modèle RBAC pour gérer les contrôles d'accès en utilisant les vues d'autorisations. Autrement dit, les vues d'autorisations peuvent être définies pour chaque utilisateur ou pour chaque rôle.

6.8 Confidentialité des vues matérialisées

Nous présentons dans cette section les approches proposées pour pallier aux problèmes de l'administration des contrôles d'accès pour assurer la confidentialité des vues matérialisées.

6.8.1 Extension du modèle Grant/Revoke

Rosenthal and Sciore [RSD99, RS00, RS01] ont proposé dans leurs travaux une théorie d'inférence pour dériver des autorisations sur les tables des entrepôts de données, dans le but de rendre les systèmes plus facile à administrer. La théorie présentée est basée sur l'extension du modèle 'Grant/Revoke' des autorisations en SQL aux systèmes avec des données dérivées et redondantes. L'extension consiste en premier lieu à définir deux types d'autorisations : Autorisation d'information et autorisation physique. La première consiste à déterminer qui est autorisé à accéder à quelle information. Cette décision est prise par l'entreprise et doit s'appliquer à toutes les vues, répliques et dérivées contenant l'information dans le système. Contrairement aux autorisations d'information, les autorisations physiques ne s'appliquent que de manière locale et déterminent qui est autorisé à accéder à quelle table physique. Par exemple : 'Les informations concernant la maladie des patients ne sont accessibles que par les médecins' est une permission d'information et 'Les médecins

peuvent requêter l'entrepôt' est une permission physique. Les auteurs justifient cette séparation d'autorisations en deux classes par une meilleure gestion lors de l'administration tout en sauvegardant la sémantique de chaque classe.

Une permission est définie comme étant un quadruplet (sujet, opération, objet, mode). Un sujet peut être un individu, un groupe, un rôle, un processus, etc. Un sujet s a une permission s'il l'a reçu explicitement, ou elle a été donnée à un ancêtre dans une hiérarchie de groupe ou de rôle. Un objet (aussi appelé table) appartient à un certain schéma, dans un certain système, soit une source ou un entrepôt. Il peut être une relation, un cube OLAP, ou bien une cellule dans une table, il peut être virtuel ou matérialisé. L'ensemble des opérations autorisées est le même que les opérations effectuées sous SQL. Si un sujet s a la permission lecture sur une table t, alors s est autorisé à accéder à toutes les données de t. Le mode d'une permission est soit physique ou d'information.

La deuxième principale extension proposée est la possibilité de déterminer si une requête a les autorisations requises pour être exécutée par rapport aux résultats de celle-ci et non pas par rapport à la manière dont elle est calculée. En effet, dans le standard SQL, une requête q d'un utilisateur ne peut être exécutée que si l'utilisateur a les autorisations d'accès à toutes les tables t_i mentionnées dans q. L'extension proposée permet à une requête q d'être exécutée s'il existe une requête q' équivalente à q et l'utilisateur a les autorisations d'accès à toutes les tables mentionnées dans q'. Les auteurs définissent la requête q' comme étant la requête *témoin* de q. Cette extension dépend de la capacité de réécriture du système pour déterminer l'équivalence des requêtes. À partir de cette définition, une permission (s, op, t, $Mode$) est définie que s'il existe une requête q définie sur $t_1, ..., t_n$ équivalente à t et chaque permission (s, op, t_i, $Mode$) a été donnée explicitement. q est appelé *témoin* de la permission impliquée. Les auteurs proposent une extension de la notion de requête témoin en définissant la permission $within - view$. Cette dernière est représentée comme étant un quintuplet (sujet, opération, objet, mode, vue). Si $p = (s,\ read,\ t,\ m,\ v)$ est une permission de l'information, alors le sujet s a le droit de lire les valeurs de la table t qui contribue au calcul de la vue v. Si p est une permission physique, alors le sujet s est autorisé à exécuter une requête qui accède physiquement à t mais seulement dans le but de calculer v.

Nous remarquons que la notion d'équivalence est très importante dans cette théorie. Prenons l'exemple suivant : Nous définissons une table dérivée t à partir de deux tables t_1 et t_2. Ces deux dernières ont respectivement les permissions d'accès (s, op, t_1, m, v_1) et (s, op, t_2, m, v_2). Dans le but de dériver une permission sur la table t, il faudrait trouver une requête q *quivalente* à t, q étant définie sur t_1 et t_2 qui contribue au calcul des vues v_1 et v_2 respectivement. Le système conclura qu'il n y a aucune permission d'accès à t si le mécanisme d'inférence n'arrive pas à calculer la requête équivalente, et cela même si l'utilisateur a le droit d'accéder à une sous partie de la table dérivée t.

L'approche proposée reste abstraite par rapport aux propriétés et à l'efficacité de la théorie d'inférence. Notons également qu'il est possible de déterminer uniquement si l'utilisateur a la permission d'accéder aux tables dérivées (en se basant sur les permissions explicites). Dans notre approche, nous proposons d'aller encore plus loin en déterminant à quelle partie de la table dérivée l'utilisateur a le droit d'accéder.

6.8.2 Réécriture de requêtes

Cuzzocrea et al. [CHG10] ont considéré le problème de comment assurer la sécurité des vues matérialisées en se basant sur les règles de contrôle d'accès de base. Ils ont proposé une nouvelle approche afin de sélectionner l'ensemble des règles de contrôle d'accès appropriées en se basant sur les algorithmes de réécritures. Le travail que nous avons réalisé dans cette thèse s'inspire de leurs travaux et a comme principal but de les compléter. Les auteurs proposent un algorithme qui, à partir (1) d'une base de données D, (2) d'un ensemble de règles de contrôle d'accès R, et (3) d'un ensemble de requêtes Q sur D définissant les vues matérialisées V ; sélectionne à partir de R, un sous ensemble Z pertinent pour assurer la sécurité de V. L'ensemble Z sélectionné est alors utilisé par un administrateur pour définir les règles de contrôle d'accès sur V. Nous souhaitons éviter cette tâche, qui peut tout aussi être ingérable pour un humain, si l'ensemble des règles sélectionné reste d'une grande taille. Notre objectif est alors de proposer une approche qui permet d'automatiser complètement le processus de génération des règles de contrôle d'accès à V. Par ailleurs, les auteurs utilisent dans leur approche l'algorithme de réécri-

ture de requêtes de base. Nous avons démontré qu'il est nécessaire d'adapter l'algorithme aux spécificités de nos besoins pour garantir une maximalité de résultats.

Chapitre 7

Évaluation

Sommaire

7.1 Introduction

Dans ce chapitre, une évaluation des performances de notre approche est effectuée. Nous n'avons besoin pour cela que du schéma d'une base de données et non de son extension et seulement des définitions des vues [1] afin de générer l'ensemble des vues d'autorisations nécessaires pour assurer la confidentialité des données au niveau des vues matérialisées.

N'ayant malheureusement pas à disposition des jeux de données réels pour évaluer les performances de notre approche, nous proposons un générateur de définitions de vues à partir d'un schéma de base de données. Le schéma des relations de base est aussi généré de manière synthétique. Ce générateur nous permet ainsi de contrôler plusieurs paramètres dans la définition des vues. Les résultats expérimentaux sont calculés sur la moyenne de plusieurs

1. Vues d'autorisations et vues matérialisées.

exécutions avec les mêmes paramètres (entre 50 et 100 exécutions pour chaque configuration). Nos principales évaluations portent sur le temps d'exécution ainsi que sur le nombre de vues générées par l'approche.

Nos expérimentations ont été menées sur une machine Intel Core 2 DUO 3GHz de processeur avec une mémoire de 4GB. Nos algorithmes ont été implémentés en java sous l'environnement NetBeans IDE 7.0.1.

7.2 Prototype

7.2.1 Générateur de jeux de données

- **Générateur de relations de base :** Il s'agit de générer le schéma de la base de données. Le nombre de relations ainsi que l'arité de chaque relation.
- **Générateur de définitions de vues :** Qu'il s'agisse des définitions des vues d'autorisations ou matérialisées le principe est le même. Une vue retourne un sous ensemble d'attributs obtenu à partir d'une relation ou d'une jointure entre deux ou plusieurs relations de base. Nous supposons dans notre implémentation, et sans perte de généralité, qu'une vue n'est définie que sur les relations de base. En effet, si une vue est définie à partir d'une autre vue, il suffira de remplacer cette dernière par son expansion pour obtenir une définition sur les relations de base uniquement. À partir de là, il est possible de gérer plusieurs paramètres : (1) le nombre d'atomes dans les vues, (2) le nombre de variables distinguées (3) le nombre de variables de jointures et (4) le nombre de fois qu'une relation puisse apparaitre dans les définitions des vues.

7.2.2 Algorithme $\mathcal{H}MiniCon^{+}$

Nous nous sommes basés sur le code source de Rachel Pottinger [PL00] pour la mise en place de notre prototype. En plus de l'algorithme MiniCon, un parseur de règles Datalog a été mis en place dans son code. Cela nous permet d'avoir un cadre de travail complet pour implémenter notre approche. Comme il a été mentionné dans le chapitre 3 (page 27), la première étape a été d'apporter les modifications nécessaires au processus de réécriture, à

savoir : (1) supprimer la condition de vérification des variables de têtes dans l'étape de formation des MCDs et (2) ajouter à la tête des réécritures les variables nouvellement introduites par le processus de réécriture. Nous avons implémenté l'algorithme de subsomption proposé dans [CGM90] pour vérifier la condition d'arrêt du processus de réécriture, c.-à-d. si la réécriture obtenue après l'application de l'algorithme $\mathcal{H}MiniCon$ contient la requête. La figure 7.1 présente la description générale de l'approche. Afin d'éviter de parcourir toutes les réécritures générées de l'arbre, nous proposons d'élaguer l'arbre au fur et à mesure de la génération des réécritures.

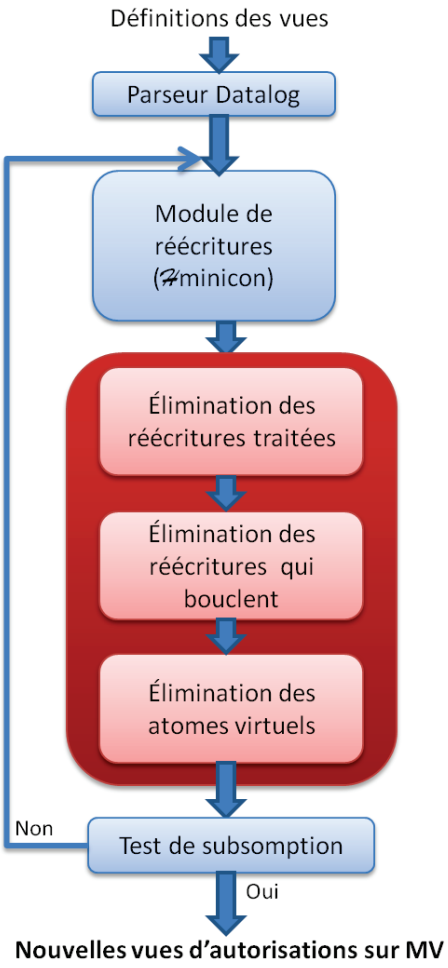

FIGURE 7.1 – Description générale de l'approche.

7.2.3 Élagage de l'arbre de réécritures

Comme montré sur la figure 7.1, l'étape d'élagage de l'arbre de réécritures est faite en trois parties :

- **Élimination des réécritures traitées** : Cette étape est importante pour réduire le nombre de réécritures traitées par le prototype. En effet, nous avons remarqué qu'une même réécriture pouvait être générée de différentes manières, c.-à-d. présente à plusieurs endroits dans l'arbre de réécritures. Dans le but de ne pas refaire le même travail plusieurs fois, nous proposons de détecter les réécritures déjà traitées et de les élaguer de l'arbre. Pour cela, nous nous basons sur le concept du *prefix tree* [2] pour stocker efficacement les réécritures générées et par la même occasion, rechercher, de façon optimale, si une réécriture a déjà été générée. L'idée est de stocker l'ensemble des atomes de chaque réécriture.
  Pour la phase de stockage, chaque réécriture doit être au préalable normalisée pour pouvoir détecter si elle a déjà été traitée. En effet, il est important de noter que pour deux requêtes données q_1 et q_2, si $normalisation(q_1) = normalisation(q_2)$ alors $q_1 \equiv q_2$. Pour cela, les atomes de chaque réécriture sont classés par ordre [3]. Ensuite, les termes sont normalisés dans l'ordre de leur apparition dans le corps de la réécriture. Il s'agira ici de remplacer chaque variable par une nouvelle variable fraiche qui n'apparait nulle part ailleurs tout en respectant l'apparition multiple d'une même variable. Autrement dit, si la variable x apparait plusieurs fois dans la requête, alors elle doit être remplacée par la même variable partout.
  Après l'étape de normalisation, la dernière étape consiste à traiter la tête de la réécriture. Étant donné que chaque réécriture a son propre nom, nous proposons de définir un atome avec le prédicat t et les termes sont les variables distinguées. Cet atome est ajouté à la fin de la liste des atomes du corps de la réécriture. Prenons l'exemple suivant pour illustrer l'étape de normalisation.

  Exemple 23 Supposons la réécriture suivante :

2. http ://whiteboxcomputing.com/java/prefix_tree/
3. Ordre alphabétique par rapport aux prédicats.

$rw_3(y_1, y_5) \leftarrow r_3(y_1, y_3), r_2(y_1, y_5), r_5(y_2, y_4).$

La première étape consiste à ordonner les atomes par ordre alphabétique par rapport aux prédicats. Nous obtenons comme résultat la requête suivante :

$rw_3(y_1, y_5) \leftarrow r_2(y_1, y_5), r_3(y_1, y_3), r_5(y_2, y_4).$

La requête est ensuite normalisée en normalisant les termes dans l'ordre de leur apparition dans le corps de la requête :

$rw_3(x_1, x_2) \leftarrow r_2(x_1, x_2), r_3(x_1, x_3), r_5(x_4, x_5).$

La liste d'atomes normalisés à insérer dans le prefix tree est la suivante : $[r_2(x_1, x_2), r_3(x_1, x_3), r_5(x_4, x_5), t(x_1, x_2)]$

L'arbre présenté sur la figure 7.2 représente un exemple d'un prefix tree. Pour vérifier si une réécriture a déjà été traitée, il faut vérifier si la liste correspondante à la réécriture normalisée est déjà présente dans le prefix tree. Pour cela, il suffit d'insérer, en respectant l'ordre, les éléments de la liste. Si la suite d'éléments existe déjà dans le prefix tree, ceci implique que la réécriture a déjà été traitée et il faut donc l'éliminer du processus de réécriture. Sinon, insérer la liste des atomes dans le prefix tree et traiter la réécriture. ◊

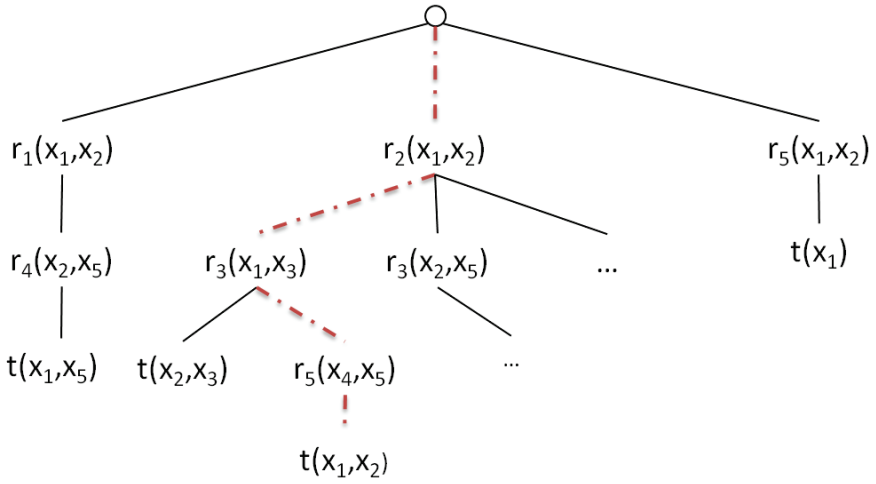

FIGURE 7.2 – Exemple d'un prefix tree.

- **Élimination des réécritures qui bouclent** : Nous avons mis en place la gestion de l'historique des atomes. Il est possible de découvrir quelle réécriture générera une boucle dans le processus de réécriture. Il suffit de détecter les atomes qui ont dans leur historique deux fois la même configuration. Comme indiqué dans le chapitre 4 (page 43) nous ne traitons pas ce cas. Nous retournons à l'administrateur la combinaison de vues qui est la source du cas de boucle.
- **Élimination des atomes virtuels** : Pour les réécritures non éliminées (après les deux étapes), il est encore possible d'alléger le processus de réécriture, en éliminant cette fois-ci les atomes correspondant aux noeuds virtuels. Nous avons démontré dans le chapitre 4, qu'en éliminant ces atomes, la requête qui en découle est équivalente à la requête d'origine. Nous rappelons qu'un atome correspond à un noeud virtuel s'il existe un autre atome avec le même historique et qui a été généré directement.

7.3 Évaluation des performances de l'approche

Nous proposons pour l'évaluation du prototype, d'étudier différents cas concernant les définitions des vues car la taille de l'arbre de réécritures en dépend. En effet, plus les relations apparaissent plusieurs fois dans les définitions des vues, plus la combinatoire pour la génération des réécritures est conséquente. Nous proposons d'étudier trois cas des définitions des vues par rapport à l'apparition des relations : (1) Sans interaction, (2) Faible interaction et (3) Forte interaction. Nous étudions dans chaque configuration le temps d'exécution et calculons le nombre de vues retournées par le prototype. Pour chaque configuration, nous calculons la moyenne de plusieurs exécutions [4].

7.3.1 Aucune interaction entre les vues

Nous étudions en premier lieu les performances de l'approche dans le cas où les définitions des vues ne partagent aucune relation en commun. La figure 7.3 montre le temps d'exécution en secondes dans 3 différentes configurations : variation du nombre de vues d'autorisations AV (10 ;50 ;100) et pour chaque cas, variation du nombre de vues matérialisées (de 10 vues jusqu'à

4. Entre 50 et 100 exécutions.

100). Nous remarquons sans surprise que le temps d'exécution est très faible. En effet, étant donné que les relations n'apparaissent pas dans plusieurs vues, le nombre de réécritures générées à chaque étape n'est pas très important. La figure 7.4 présente le nombre de vues générées en moyenne pour les différentes configurations. Nous remarquons que le nombre de vues générées est proportionnel à la taille des deux ensembles de vues.

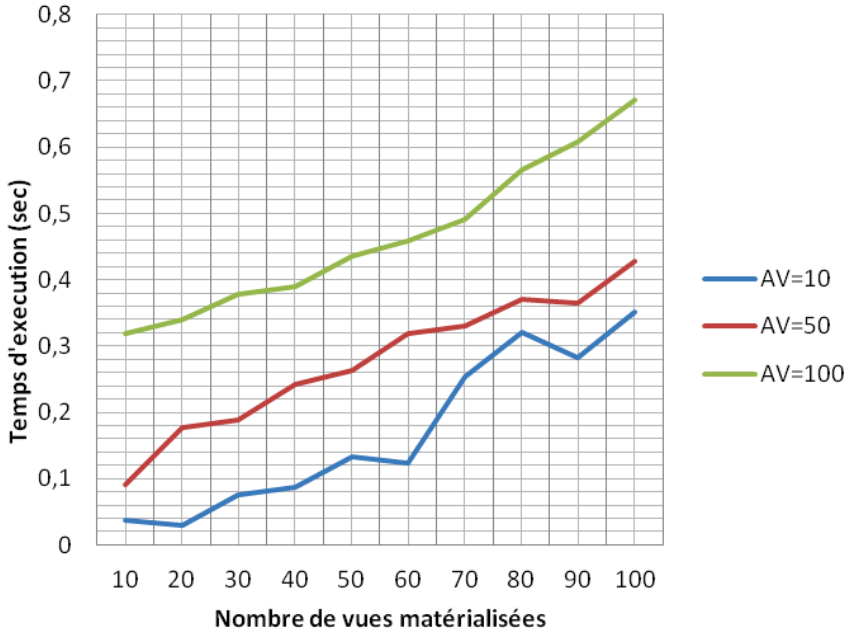

FIGURE 7.3 – Temps d'exécution par rapport au nombre de vues dans les deux ensembles. -Aucune interaction-

7.3.2 Faible interaction entre les vues

Pour la deuxième partie des tests, nous introduisons une faible interaction entre les vues c.-à-d. un sous ensemble des vues partagent entre elles certaines relations. Nous avons choisi de faire apparaitre la même relation dans 20% des vues d'un ensemble. Nous remarquons sur la figure 7.6 que cette interaction n'affecte que modérément le nombre de vues générées, par contre, le temps d'exécution (figure 7.5) est plus conséquent. Ceci s'explique par le nombre plus important de réécritures que l'algorithme génère dans le cas où la relation choisie, qui apparait dans plusieurs vues, apparait dans la requête de départ.

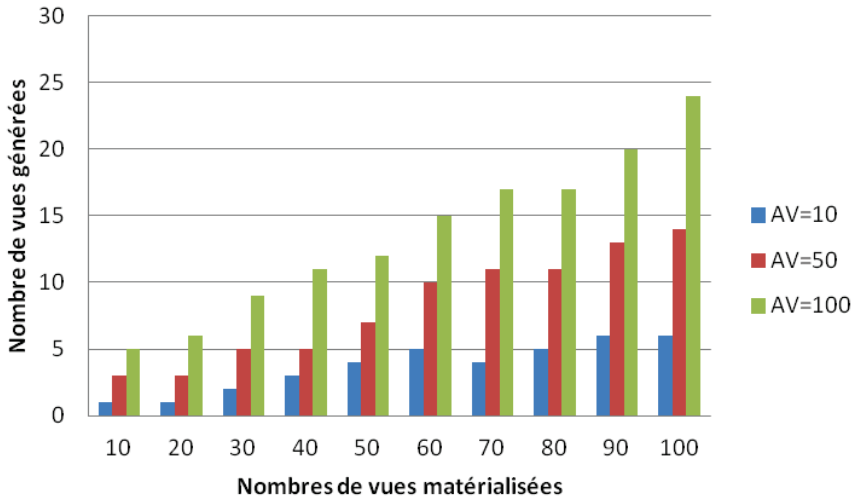

FIGURE 7.4 – Nombre de vues générées par rapport au nombre de vues dans les deux ensembles. -Aucune interaction-

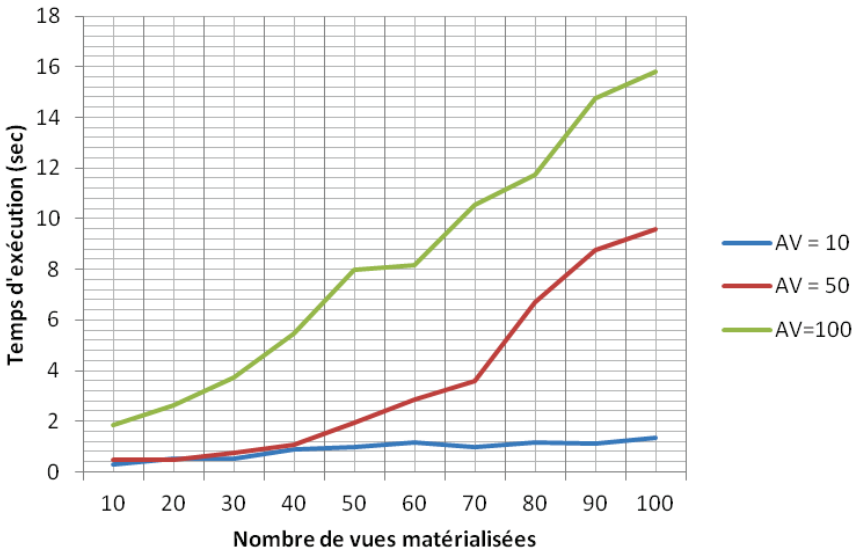

FIGURE 7.5 – Temps d'exécution par rapport au nombre de vues dans les deux ensembles. -Interaction 20%-

7.3.3 Forte interaction entre les vues

Nous étudions dans cette partie le cas où plus de 50% des relations peuvent apparaitre dans différentes définitions de vues. L'apparition récurrente de relations dans plus de 50% des vues implique une génération très importante de

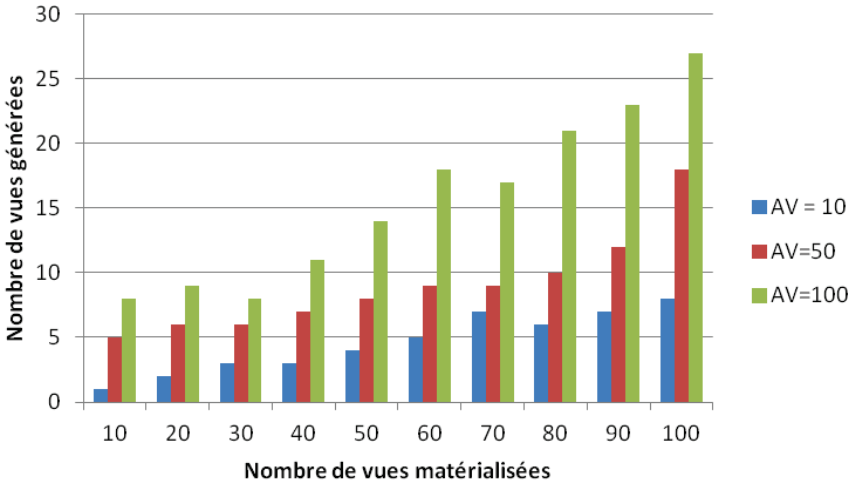

FIGURE 7.6 – Nombre de vues générées par rapport au nombre de vues dans les deux ensembles. -Interaction 20%-

réécritures due à la grande combinatoire générée pour chaque requête. Pour ce cas, nous ne pouvons pas traiter un grand nombre de vues à cause de la complexité réelle de l'algorithme. Nous proposons donc d'étudier sur un petit échantillon de vues, le comportement de notre algorithme. Les figures 7.7 et 7.8 nous montrent que pour deux ensembles contenant chacun une dizaine de vues et en variant le nombre de vues partageant la même relation (de 30% à 70%), le temps d'exécution et le nombre de vues générées sont très élevés comparés aux cas précédents. Ce résultat est justifié. En effet, réécrire une requête en utilisant un ensemble de vues avec une forte interaction revient à calculer un produit cartésien entre presque toutes les vues et ceci pour chaque requête et plus l'arbre de réécritures correspondant est profond, plus les réécritures contiennent un nombre conséquent d'atomes et donc de plus en plus de réécritures générées.

Pour le passage à l'échelle dans le cas de forte interaction entre les vues, une optimisation de l'approche, e.x., réduction des requêtes, traitement plus efficace de l'inclusion de requêtes (voir perspectives) est nécessaire pour pouvoir traiter un grand nombre de vues.

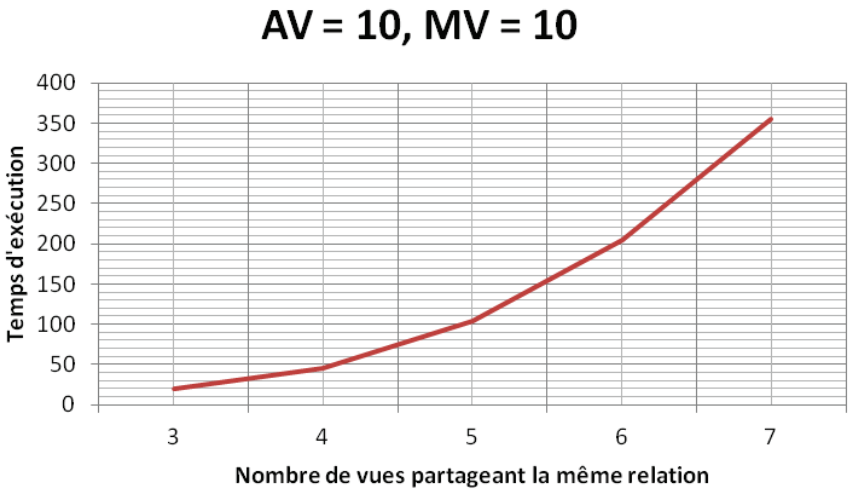

FIGURE 7.7 – Temps d'exécution par rapport au nombre de vues dans les deux ensembles. -Interaction 50%-

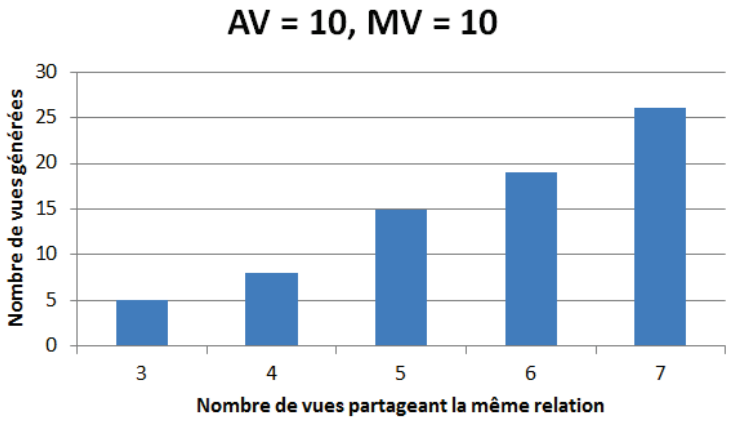

FIGURE 7.8 – Nombre de vues générées par rapport aux nombre de vues partageant la même relation. -Interaction 50%-

Chapitre 8

Conclusion et Perspectives

Sommaire

8.1 Synthèse

Dans cette thèse, nous nous sommes intéressés au problème de caractérisation des informations que l'on peut récupérer à partir de deux ensembles de vues $\mathcal{V}_1$ et $\mathcal{V}_2$. Nous nous sommes focalisés pour cela sur l'usage des algorithmes de réécriture de requêtes en utilisant des vues. Nous avons traité les requêtes conjonctives en autorisant uniquement les égalités. Pour résoudre ce problème, nous nous sommes appuyés sur (1) le langage *Datalog* pour la formalisation du problème et l'étude des propriétés théoriques et (2) l'algorithme de réécriture de requête *MiniCon*, qui représente le socle de base de notre travail. Nous avons proposé l'algorithme $\mathcal{H}MiniCon$ qui est une adaptation de *MiniCon* aux besoins du problème, à savoir, permettre la possibilité de calculer plus de réécritures (que l'algorithme de base) et par conséquent, récupérer plus d'informations. Nous avons démontré que l'enchainement de l'application de l'algorithme $\mathcal{H}MiniCon$ permet de calculer exactement la caractérisation des informations calculables à partir des deux ensembles de vues dans le cas où l'algorithme terminait.

Deux structures résultent de l'application successive de l'algorithme de réécriture adapté : (1) l'arbre de réécritures associé à chaque requête, qui représente les différentes réécritures produites par l'algorithme et (2) l'arbre d'atomes associé à chaque branche de l'arbre de réécritures, cette structure

permet d'exhiber les informations lors de la génération de chaque réécriture. À partir de ces deux structures, nous avons étudié les conditions de terminaison de notre approche. Nous avons pu définir dans quel cas l'algorithme $\mathcal{H}MiniCon^{+}$ terminait et nous avons démontré que dans ce cas-là, notre approche est maximale. Nous avons également exhibé la configuration qui risquait de générer des arbres de réécritures infinis et par conséquent, aucun moyen de déterminer ni la terminaison de l'approche ni sa maximalité.

Le cadre applicatif de cette thèse est la confidentialité des données au niveau des vues matérialisées. Dans le cas de grosses organisations, la gestion des données à travers les vues matérialisées est très répondue. Assurer la confidentialité des données en présence de vues matérialisées est aussi important que la confidentialité des données au niveau des tables de bases. Parmi les modèles proposées pour assurer la confidentialité des données, nous avons choisi les vues d'autorisations qui permettent une définition de règles d'accès à grains fins et basé sur le contenu. En appliquant notre approche dans le domaine de la sécurité, nous déterminons ainsi l'ensemble des vues d'autorisations qui assureront la confidentialité des données au niveau des vues matérialisées.

Nous avons implémenté l'approche sous forme d'un prototype. Les expérimentations réalisées sur des jeux de données synthétiques démontrent que dans le cas d'une faible interaction entre les vues, la complexité réelle de l'approche est moins importante que la complexité théorique présentée. En effet, à chaque étape de réécriture, seulement un sous ensemble de vues sera considéré comme contributif à la réécriture, ce qui par conséquent, créé une moindre combinatoire dans la génération des réécritures. Dans le cas où il existe une forte interaction entre les vues, nos expérimentations démontrent que le temps d'exécution et la mémoire consommée sont beaucoup plus conséquents. Ce résultat est prévisible en raison du nombre de combinaisons possible entre les vues pour chaque réécriture dans l'arbre.

8.2 Perspectives

Nous avons défini, à travers notre approche, un cadre de travail de base pour répondre à la problématique de caractérisation d'informations calculables à partir de deux ensembles de vues. Dans le but d'optimiser notre approche nous proposons différentes perspectives :

- Il est connu que le problème d'inclusion de requêtes est NP-complet. Nous avons utilisé dans notre approche l'algorithme naïf pour déterminer si une requête est contenue dans une autre. Plusieurs algorithmes sont proposés dans la littérature et peuvent être utilisés dans notre approche pour déterminer de manière efficace l'inclusion de requêtes. À titre d'exemple, Chekuri et al. [CR00] présentent un algorithme qui teste l'inclusion en un temps polynomial pour la classe de requêtes acycliques.
- Nous avons proposé dans notre approche de remplacer les requêtes à réécrire par des requêtes équivalentes en éliminant les atomes virtuels. Il est possible d'optimiser davantage le processus de réécriture en appliquant les algorithmes qui déterminent, à partir d'une requête, la requête minimale équivalente [CM77].

Nous avons mis en évidence les cas où l'algorithme peut ne pas terminer à cause de boucles infinies. Une des perspectives importantes de notre travail est l'étude de ces boucles. Déterminer si la détection des boucles est une condition nécessaire et suffisante pour caractériser la non terminaison de l'algorithme. Étudier les caractéristiques des vues non traitées dues au fait qu'on force l'arrêt de l'algorithme.

Pour ce premier travail, nous avons traité les requêtes conjonctives en autorisant uniquement les égalités. D'un point de vue théorique, diverses perspectives peuvent être envisagées dans la continuité de ce travail. Les résultats présentés dans cette thèse peuvent être étendus au cas de requêtes conjonctives avec prédicats de comparaisons [ALM06]. Dans ce cas, nous pouvons déjà conclure que l'approche restera correcte. Néanmoins, un travail sur l'étude de la terminaison est nécessaire. Notre étude de terminaison se base sur la finitude du nombre d'atomes ayant des historiques différents. Cette condition n'est plus forcément vérifiée dans le cadre des requêtes conjonctives avec pré-

dicats de comparaison. Une autre extension peut être considérée pour traiter les vues et les requêtes utilisant la négation [AP06].

Plus spécifiquement au cadre applicatif, les vues matérialisées sont très utilisées dans les entrepôts de données pour des raisons d'optimisation. Elles permettent de calculer et de sauvegarder des données agrégées. Une extension de notre travail est possible en étudiant les requêtes avec les fonctions d'agrégats. L'approche peut donc être adaptée en utilisant les algorithmes de réécriture de requêtes avec fonctions d'agrégats [CNS06, SDJL96]. Néanmoins, certaines questions doivent être posées sur la position à prendre d'un point de vue sécurité. En effet, il existe différentes visions. Par exemple, si dans les vues d'autorisations de base un utilisateur n'a pas le droit de voir certains tuples d'une table. Est-ce que le fait de voir la moyenne ou la somme de ces tuples peut divulguer certaines informations ? Dans quel cas, est-il permis à l'utilisateur d'accéder aux données agrégées sans qu'il ait divulgation d'informations ? Plusieurs approches pourraient être proposées pour répondre aux différentes visions.

Concernant le prototype développé, un module de conversion de requêtes SQL vers Datalog (et de Datalog vers SQL) doit être mis en place pour rendre plus facile son utilisation. Nous avons proposé quelques méthodes d'optimisation de l'approche comme l'utilisation des *prefix trees* pour éliminer les réécritures déjà traitées. Il est possible d'améliorer davantage les performances en parallélisant le processus de réécriture de requêtes.

Bibliographie

[ABC+76] Morton M. Astrahan, Mike W. Blasgen, Donald D. Chamberlin, Kapali P. Eswaran, Jim Gray, Patricia P. Griffiths, W. Frank King III, Raymond A. Lorie, Paul R. McJones, James W. Mehl, Gianfranco R. Putzolu, Irving L. Traiger, Bradford W. Wade, and Vera Watson. System r : Relational approach to database management. *ACM Trans. Database Syst.*, 1(2) :97–137, 1976. (Cited on page 76.)

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995. (Cited on pages 5 and 11.)

[ALM06] Foto N. Afrati, Chen Li, and Prasenjit Mitra. Rewriting queries using views in the presence of arithmetic comparisons. *Theor. Comput. Sci.*, 368(1-2) :88–123, 2006. (Cited on page 103.)

[AP06] Foto N. Afrati and Vassia Pavlaki. Rewriting queries using views with negation. *AI Commun.*, 19(3) :229–237, 2006. (Cited on page 104.)

[BBF00] Elisa Bertino, Piero A. Bonatti, and Elena Ferrari. Trbac : a temporal role-based access control model. In *ACM Workshop on Role-Based Access Control*, pages 21–30, 2000. (Cited on page 79.)

[BBFS98] Elisa Bertino, Claudio Bettini, Elena Ferrari, and Pierangela Samarati. An access control model supporting periodicity constraints and temporal reasoning. *ACM Trans. Database Syst.*, 23(3) :231–285, 1998. (Cited on page 77.)

[BF97] Elisa Bertino and Elena Ferrari. Administration policies in a multipolicy autorization system. In *DBSec*, pages 341–355, 1997. (Cited on page 76.)

[BLP76] E. D. Bell and J. L. La Padula. Secure computer system : Unified exposition and multics interpretation, 1976. (Cited on page 78.)

[BS05] Elisa Bertino and Ravi Sandhu. Database security-concepts, approaches, and challenges. *IEEE Trans. Dependable Secur. Comput.*, 2(1) :2–19, 2005. (Cited on pages 74 and 81.)

[CBHB09] Leonardo Weiss Ferreira Chaves, Erik Buchmann, Fabian Hueske, and Klemens Böhm. Towards materialized view selection for distributed databases. In *EDBT*, pages 1088–1099, 2009. (Cited on page 2.)

[CGM90] Upen S. Chakravarthy, John Grant, and Jack Minker. Logic-based approach to semantic query optimization. *ACM Trans. Database Syst.*, 15(2) :162–207, 1990. (Cited on pages 35 and 93.)

[CHG10] Alfredo Cuzzocrea, Mohand-Said Hacid, and Nicola Grillo. Effectively and efficiently selecting access control rules on materialized views over relational databases. In *IDEAS*, pages 225–235, 2010. (Cited on page 88.)

[CKPS95] Surajit Chaudhuri, Ravi Krishnamurthy, Spyros Potamianos, and Kyuseok Shim. Optimizing queries with materialized views. In *ICDE*, pages 190–200, 1995. (Cited on page 16.)

[CM77] Ashok K. Chandra and Philip M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *STOC*, pages 77–90, 1977. (Cited on page 103.)

[CM04] Frédéric Cuppens and Alexandre Miège. Adorbac : an administration model for or-bac. *Comput. Syst. Sci. Eng.*, 19(3), 2004. (Cited on page 79.)

[CNS06] Sara Cohen, Werner Nutt, and Yehoshua Sagiv. Rewriting queries with arbitrary aggregation functions using views. *ACM Trans. Database Syst.*, 31(2) :672–715, 2006. (Cited on page 104.)

[CR00] Chandra Chekuri and Anand Rajaraman. Conjunctive query containment revisited. *Theor. Comput. Sci.*, 239(2) :211–229, 2000. (Cited on page 103.)

[DBCP07] Maria Luisa Damiani, Elisa Bertino, Barbara Catania, and Paolo Perlasca. Geo-rbac : A spatially aware rbac. *ACM Trans. Inf. Syst. Secur.*, 10(1), 2007. (Cited on page 79.)

[DG97] Oliver M. Duschka and Michael R. Genesereth. Answering recursive queries using views. In *PODS*, pages 109–116, 1997. (Cited on page 21.)

[dVFJ+08] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. Assessing query privileges via safe and efficient permission composition. In *ACM Conference on Computer and Communications Security*, pages 311–322, 2008. (Cited on page 80.)

[EC91] CommissionoftheEuropeanCommunities EC. *Information Technology Security Evaluation Criteria (ITSEC)*. Commission of the European Communities, 06 1991. (Cited on page 73.)

[FCG04] Wenfei Fan, Chee Yong Chan, and Minos N. Garofalakis. Secure xml querying with security views. In *SIGMOD Conference*, pages 587–598, 2004. (Cited on page 75.)

[FK09] David F. Ferraiolo and D. Richard Kuhn. Role-based access controls. *CoRR*, abs/0903.2171, 2009. (Cited on page 79.)

[FSG+01] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3) :224–274, 2001. (Cited on page 75.)

[GD72] G. Scott Graham and Peter J. Denning. Protection : principles and practice. In *AFIPS Spring Joint Computing Conference*, pages 417–429, 1972. (Cited on page 75.)

[GMR95] Ashish Gupta, Inderpal Singh Mumick, and Kenneth A. Ross. Adapting materialized views after redefinitions. In *SIGMOD Conference*, pages 211–222, 1995. (Cited on page 17.)

[GSC+09] Benoît Groz, Slawomir Staworko, Anne-Cécile Caron, Yves Roos, and Sophie Tison. Xml security views revisited. In *DBPL*, pages 52–67, 2009. (Cited on page 75.)

[GW76] Patricia P. Griffiths and Bradford W. Wade. An authorization mechanism for a relational database system. *ACM Trans. Database Syst.*, 1(3) :242–255, 1976. (Cited on page 76.)

[Hal01] Alon Y. Halevy. Answering queries using views : A survey. *VLDB J.*, 10(4) :270–294, 2001. (Cited on page 17.)

[HRU76] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8) :461–471, 1976. (Cited on page 75.)

[JLS76] Anita K. Jones, Richard J. Lipton, and Lawrence Snyder. A linear time algorithm for deciding security. In *FOCS*, pages 33–41, 1976. (Cited on page 75.)

[Kuh97] D. Richard Kuhn. Mutual exclusion of roles as a means of implementing separation of duty in role-based access control systems. In *ACM Workshop on Role-Based Access Control*, pages 23–30, 1997. (Cited on page 79.)

[Lam74] Butler W. Lampson. Protection. *Operating Systems Review*, 8(1) :18–24, 1974. (Cited on page 75.)

[LRO96] Alon Y. Levy, Anand Rajaraman, and Joann J. Ordille. Querying heterogeneous information sources using source descriptions. In *VLDB*, pages 251–262, 1996. (Cited on pages 16 and 19.)

[Mot89] Amihai Motro. An access authorization model for relational databases based on algebraic manipulation of view definitions. In *ICDE*, pages 339–347, 1989. (Cited on page 2.)

[OGM08] Lars E. Olson, Carl A. Gunter, and P. Madhusudan. A formal framework for reflective database access control policies. In *ACM Conference on Computer and Communications Security*, pages 289–298, 2008. (Cited on pages 2 and 75.)

[Ora02] Oracle. The virtual private database in oracle9ir2 : An oracle techni- cal white paper. *Available at http ://www.oracle.com*, 2002. (Cited on pages 77, 78 and 83.)

[Per94] Günther Pernul. Database security. *Advances in Computers*, 38 :1–72, 1994. (Cited on pages 76 and 78.)

[PL00] Rachel Pottinger and Alon Y. Levy. A scalable algorithm for answering queries using views. In *VLDB*, pages 484–495, 2000. (Cited on pages 22, 25 and 92.)

[PP01] Torsten Priebe and Günther Pernul. A pragmatic approach to conceptual modeling of olap security. In *In Proc. ER*, pages 311–324. Springer-Verlag, 2001. (Cited on page 2.)

[Qia96] Xiaolei Qian. Query folding. In *ICDE*, pages 48–55, 1996. (Cited on page 21.)

[RMSR04] Shariq Rizvi, Alberto O. Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *SIGMOD Conference*, pages 551–562, 2004. (Cited on pages 2, 77, 82, 84 and 85.)

[RS00] Arnon Rosenthal and Edward Sciore. View security as the basis for data warehouse security. In *DMDW*, page 8, 2000. (Cited on page 86.)

[RS01] Arnon Rosenthal and Edward Sciore. Administering permissions for distributed data : Factoring and automated inference. In *DBSec*, pages 91–104, 2001. (Cited on page 86.)

[RSD99] Arnon Rosenthal, Edward Sciore, and Vinti Doshi. Security administration for federations, warehouses, and other derived data. In *DBSec*, pages 209–223, 1999. (Cited on page 86.)

[San93] Ravi S. Sandhu. Lattice-based access control models. *IEEE Computer*, 26(11) :9–19, 1993. (Cited on page 75.)

[SCFY96] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2) :38–47, 1996. (Cited on pages 77 and 79.)

[SDJL96] Divesh Srivastava, Shaul Dar, H. V. Jagadish, and Alon Y. Levy. Answering queries with aggregation using views. In *VLDB*, pages 318–329, 1996. (Cited on page 104.)

[SG00] Jürgen Steger, Holger Günzel, and Andreas Bauer 0004. Identifying security holes in olap applications. In Bhavani M. Thuraisingham, Reind P. van de Riet, Klaus R. Dittrich, and Zahir Tari, editors, *DBSec*, volume 201 of *IFIP Conference Proceedings*, pages 283–294. Kluwer, 2000. (Cited on page 2.)

[SW74] Michael Stonebraker and Eugene Wong. Access control in a relational data base management system by query modification. In *Proceedings of the 1974 annual conference - Volume 1*, ACM '74, pages 180–186, New York, NY, USA, 1974. ACM. (Cited on page 78.)

[SY78] Yehoshua Sagiv and Mihalis Yannakakis. Equivalence among relational expressions with the union and difference operation. In *VLDB*, pages 535–548, 1978. (Cited on page 16.)

[TS97] Roshan K. Thomas and Ravi S. Sandhu. Task-based authorization controls (tbac) : A family of models for active and enterprise-oriented autorization management. In *DBSec*, pages 166–181, 1997. (Cited on pages 77 and 79.)

[TS99] Dimitri Theodoratos and Timos Sellis. Dynamic data warehouse design. In *1st Int. Conf. on DaWak '99*, pages 1–10. Springer-Verlag, 1999. (Cited on page 2.)

[TSI94] Odysseas G. Tsatalos, Marvin H. Solomon, and Yannis E. Ioannidis. The gmap : A versatile tool for physical data independence. In *VLDB*, pages 367–378, 1994. (Cited on page 16.)

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In *ICDT*, pages 19–40, 1997. (Cited on page 17.)

[Win] Oracle database : Data warehousing guide. http://docs.oracle.com/cd/B28359_01/server.111/b28313.pdf. Accessed : 2007-09. (Cited on page 3.)

[WMT02] Junhu Wang, Michael Maher, and Rodney Topor. Rewriting unions of general conjunctive queries using views. In *Proc. Conf. on Extending Database Technology, LNCS 2287*, 2002. (Cited on page 37.)

[WYL+07] Qihua Wang, Ting Yu, Ninghui Li, Jorge Lobo, Elisa Bertino, Keith Irwin, and Ji-Won Byun. On the correctness criteria of fine-grained access control in relational databases. In *VLDB '07 : Proceedings of the 33rd international conference on Very large data bases*, pages 555–566. VLDB Endowment, 2007. (Cited on page 2.)

[YKL97] Jian Yang, Kamalakar Karlapalem, and Qing Li. Algorithms for materialized view design in data warehousing environment. In *VLDB*, pages 136–145, 1997. (Cited on page 2.)

[YL87] H. Z. Yang and Per-$\mathcal{R}$Ake Larson. Query transformation for psj-queries. In Peter M. Stocker, William Kent, and Peter Hammersley, editors, *VLDB'87, Proceedings of 13th International Conference on Very Large Data Bases, September 1-4, 1987, Brighton, England*, pages 245–254. Morgan Kaufmann, 1987. (Cited on page 16.)

Liste des symboles