Etude et implantation de l'extraction de requêtes fréquentes dans les bases de données multidimensionnelles.

Cheikh Tidiane Dieng

Département des Sciences Informatiques **École Doctorale Sciences et Ingénierie**
UFR des Sciences et Techniques

ETUDE ET IMPLANTATION DE L'EXTRACTION DE REQUETES FREQUENTES DANS LES BASES DE DONNEES MULTIDIMENSIONNELLES

THÈSE

présentée et soutenue publiquement le 19 Juillet 2011

pour l'obtention du

Doctorat en Sciences de l'Université de CERGY-PONTOISE (CERGY)

(spécialité informatique)

par

Cheikh Tidiane DIENG

Composition du jury

Rapporteurs :	Jean Marc PETIT Professeur INSA-Lyon Bart GOETHALS Professeur Université d'Antwerp
Examinateurs :	Bruno CREMILLEUX Professeur Université de Caen Oumar SALL Professeur Université de Ziguinchor
Directeur de thèse:	Dominique LAURENT Professeur Université de Cergy-Pontoise
Co-Encadreur:	Tao-Yuan JEN Maitre de Conférences Université de Cergy-Pontoise

Equipe Traitement de l'Information et Systèmes (ETIS) – UMR CNRS 8051

Mis en page avec la classe thloria.

Remerciements

Je tiens à remercier très sincèrement mon directeur de thèse Dominique LAURENT d'avoir bien voulu me proposer ce sujet de thèse. Merci infiniment pour les orientations, et la rigueur scientifique. Je ne suis pas insensible à votre bonne humeur tout au long de cette thèse.

Merci aussi d'avoir accepté le principe de la thèse en alternance entre le Sénégal et la France, malgré mes tâches d'enseignant au SENEGAL. Je tiens à exprimer mes plus

vifs remerciements à M. Tao-Yuan JEN, Maitres de Conférences à l'Université de Cergy-Pontoise. Merci infiniment pour la disponibilité, la patience, les remarques et suggestions pertinentes durant toute la thèse.

J'ai beaucoup appris durant ces quatre années et j'espère vivement que cette thèse sera le début d'une collaboration scientifique continue.

Mes remerciements vont également à l'endroit de Mary Teuw NIANE, Recteur de l'Université Gaston Berger de Saint-Louis, qui a bien voulu être le co-directeur de cette thèse pour la partie sénégalaise de la co-tutelle.

Je remercie les rapporteurs : Jean-Marc PETIT, de l'INSA de Lyon (France) et Bart GOETHALS, de l'Université d'Antwerp (Belgique) et les examinateurs : M. Oumar SALL de l'Université de Ziguinchor (Sénégal), M. Bruno CREMILLEUX de l'Université de Caen (France) d'avoir consenti à juger ce travail.

Je remercie le Service de Coopération et d'Action culturelle de l'Ambassade de France au Sénégal pour son soutien financier qui m'a permis d'effectuer mes séjours de thèse en alternance entre l'ETIS de Cergy et le LANI de l'Université Gaston Berger de Saint-Louis. Je remercie également Dominique LAURENT qui m'a permis, grâce à un poste d'ATER à l'Université de Cergy-Pontoise, de poursuivre cette thèse dans de bonnes conditions matérielles à la fin de mes douze mois de bourses. Je remercie également tous mes collègues de

l'Université de Ziguinchor pour leur soutien.Plus personnellement, je remercie tous mes amis qui m'ont beaucoup soutenu : Jean Marie DEMBELE, Farra N'DIAYE, El Hadj Mamadou NDIAYE, Diene DIOUF, Khalifa GAYE, Pape Ousmane COLY, Ousmane THIARE, N'Dèye Fatou PAYE, Valentin TOMASSO, Moise BASSE, Moussa N'DIAYE, Moussa BASSEL sans oublier également tous mes amis de FACEBOOK (que j'ai beaucoup soulé durant cette thèse par mes posts).Je remercie également M'Baye SENE de

l'Université de Dakar pour son soutien durant les moments particulièrement difficiles de ma vie, merci pour tout. Enfin, je remercie mes parents et mes frères pour leur amour.

A mes parents ! ! !

Résumé

Au cours de ces dernières années, le problème de la recherche de requêtes fréquentes dans les bases de données est un problème qui a suscité de nombreuses recherches. En effet, beaucoup de motifs intéressants comme les règles d'association, des dépendances fonctionnelles exactes ou approximatives, des dépendances fonctionnelles conditionnelles exactes ou approximatives peuvent être découverts simplement, contrairement au méthodes classiques qui requièrent plusieurs transformations de la base pour extraire de tels motifs. Cependant, le problème de la recherche de requêtes fréquentes dans les bases de données relationnelles est un problème difficile car, d'une part l'espace de recherche est très grand (puisque égal à l'ensemble de toutes les requêtes pouvant être posées sur une base de données), et d'autre part, savoir si deux requêtes sont équivalentes (donc engendrant les calculs de support redondants) est un problème NP-Complet.

Dans cette thèse, nous portons notre attention sur les requêtes de type Projection-Selection-Jointure (PSJ), et nous supposons que la base de données est définie selon un schéma étoile. Sous ces hypothèses, nous définissons une relation de pré-ordre ($\leq$) entre les requêtes et nous montrons que :

1. La mesure de support est anti-monotone par rapport à $\leq$, et
2. En définissant, $q \equiv q'$ si et seulement si $q \leq q'$ et $q' \leq q$, alors toutes les requêtes d'une même classe d'équivalence ont même support.

Les principales contributions de cette thèse sont, d'une part d'étudier formellement les propriétés du pré-ordre et de la relation d'équivalence ci-dessus, et d'autre part, de proposer un algorithme par niveau de type Apriori pour rechercher l'ensemble des requêtes fréquentes d'une base de données définie sur un schéma étoile. De plus, cet algorithme a été implémenté et les expérimentations que nous avons réalisées montrent que, selon notre approche, le temps de calcul des requêtes fréquentes dans une base de données définie sur un schéma étoile reste acceptable, y compris dans le cas de grandes tables de faits.

Mots-clés: base de données, fouilles de données, requêtes, motifs, algorithme par niveau, connaissances

Abstract

The problem of mining frequent queries in a database has motivated many research efforts during the last two decades. This is so because many interesting patterns, such as association rules, exact or approximative functional dependencies and exact or approximative conditional functional dependencies can be easily retrieved, which is not possible using standard techniques. However, the problem mining frequent queries in a relational database is not easy because, on the one hand, the size of the search space is huge (because encompassing all possible queries that can be addressed to a given database), and on the other hand, testing whether two queries are equivalent (which entails redundant support computations) is NP-Complete.

In this thesis, we focus on Projection-Selection-Join (PSJ) queries, assuming that the database is defined over a star schema. In this setting, we define a pre-ordering ($q \leq q'$) between queries and we prove the following basic properties :

1. The support measure is anti-monotonic with respect to $\leq$, and
2. Defining $q \equiv q'$ if and only if $q \leq q'$ and $q' \leq q$, all equivalent queries have the same support.

The main contributions of the thesis are, on the one hand to formally study properties of the pre-ordering and the equivalence relation mentioned above, and on the other hand, to prose a level-wise, Apriori like algorithm for the computation of all frequent queries in a relational database defined over a star schema. Moreover, this algorithm has been implemented and the reported experiments show that, in our approach, runtime is acceptable, even in the case of large fact tables.

Keywords: database, datamining, query, pattern, level-wise algorithm, knowledge

Table des matières

1 INTRODUCTION **3**

2 ETAT DE L'ART **9**

2.1 Introduction 9
2.2 Le Datamining ou Fouille de données 9
2.3 Recherche de motifs fréquents 10
2.3.1 Formalisme général de recherche de motifs intéressants 10
2.3.2 Algorithme général de recherche de motifs intéressants 11
2.4 Recherche d'itemsets et règles fréquentes 12
2.5 Recherche de motifs dans les bases de données relationnelles 13
2.5.1 Formalisme de la découverte de motifs dans les bases de données relationnelles 16
2.5.2 Les critères de qualité 19
2.6 Algorithmes de recherches de motifs dans le cas multi-relationnel 19
2.6.1 Approche Programmation Logique Inductive : `WARMR` 20
2.6.2 Approche Programmation Logique Inductive : `WARMER` 23
2.6.3 Approche Base de données : `Conqueror` 28
2.6.4 Approche Base de données : `IncMiner` 37
2.7 Conclusion 43

3 CONTRIBUTION **45**

3.1 Introduction 45
3.2 Modèle Formel 46
3.2.1 Introduction 46
3.2.2 Requêtes 46
3.3 Recherche de requêtes fréquentes dans les bases de données définies sur un schéma étoile 49
3.3.1 Introduction 49
3.3.2 Relation de pré-ordre sur les requêtes d'une base de données ayant un schéma étoile 50
3.3.3 Classes d'équivalence de requêtes 54
3.3.4 Calcul de $\mathcal{C}^*$ 57
3.4 Algorithmes 61

3.4.1 Algorithme Principal : FQF . . . 62
3.4.2 Algorithme mine . . . 62
3.4.3 Algorithme generate . . . 64
3.4.4 Algorithme prune . . . 64
3.4.5 Algorithme scan . . . 67
3.5 Complexité de FQF . . . 70
3.6 Une approche utilisant les résultats de notre contribution : Conqueror$^+$ ([99]) 71
3.6.1 Boucle de Jointure : . . . 74
3.6.2 Boucle de Projection . . . 74
3.6.3 Boucle de Sélection : . . . 75
3.6.4 Résultats expérimentaux : . . . 76
3.7 Conclusion . . . 77

4 IMPLÉMENTATION 79
4.1 Introduction . . . 79
4.2 Interface Graphique de l'Application . . . 79
4.3 Analyse et Conception . . . 80
4.3.1 Architecture générale de l'implémentation . . . 80
4.3.2 Choix de langage et de Système de Gestion de base de données . . . 80
4.4 Classes . . . 81
4.4.1 Générateur de base de données sous forme de schéma étoile . . . 82
4.4.2 Développements . . . 82
4.5 Problèmes rencontrés et Solutions proposées . . . 84
4.5.1 Problèmes rencontrés . . . 84
4.5.2 Différents cas de redondance . . . 84
4.5.3 Elimination partielle des redondances . . . 88
4.5.4 Complétude de l'implémentation de notre algorithme FQF . . . 92
4.6 Tests effectués . . . 94
4.6.1 Performances de FQF . . . 95
4.6.2 Résultats comparatifs de FQF, Conqueror et Conqueror$^+$ . . . 97
4.7 Conclusion . . . 101

5 Conclusions et Perspectives 103
5.1 Résumé de nos contributions . . . 103

5.2 Perspectives 104

Bibliographie **107**

Table des figures **115**

Appendices **117**

A Preuves des propositions **119**

A.1 Proposition 119

A.2 Preuve proposition 3.7 122

B Documentation des classes Java **123**

B.1 Diagrammes de classes 123

B.2 Diagrammes de séquences 125

B.3 Loader 129

B.4 Aux 129

B.5 Tuple 129

B.6 Saver 130

B.7 Schema 130

B.8 GenericClass 131

B.9 QueryClass 132

B.10 Frequent Queries 132

1 INTRODUCTION

Avec l'augmentation sans cesse de la capacité de stockage des ordinateurs, nous assistons au développement des moyens de collecte et de stockage des données.

Aujourd'hui, les banques, les sociétés de télécommunication, les organismes scientifiques ou entreprises de la grande distribution disposent de masses d'informations collectées et consolidées, qui deviennent très utiles pour de meilleures prises de décision. Souvent, ces masses de données contiennent des informations cachées qui peuvent être pertinentes. Ces informations pertinentes sont appelées connaissances ou motifs intéressants dans la littérature dédiée.

Nous pouvons citer quelques applications pouvant découler de la découverte de tels motifs :

- Détection de fraude à la carte bancaire ;
- Aide à l'octroi de crédit (gestion de risques) ;
- Aide à la prescription d'un traitement médical ;
- Aide au diagnostic de panne dans le domaine aéronautique ;
- Aide dans la stratégie de marketing d'une entreprise de grande distribution.

Cette liste d'applications n'est pas exhaustive, en effet, beaucoup d'autres applications mettant en jeu la recherche d'informations dans de grosses masses de données existent dans la vie courante.

Cependant, l'exploration ou l'exploitation de ces masses de données pour trouver des motifs intéressants, comme nous le verrons dans la suite, pose un certain nombre de difficultés.

C'est ainsi que beaucoup de scientifiques se sont intéressés à ce domaine appelé Fouille de données ou Datamining en Anglais.

La Fouille de données est définie comme "la découverte d'informations intéressantes, non triviales, implicites, préalablement inconnues et potentiellement utiles à partir de grandes bases de données" [60].

Cette definition, est une des premières qui traite explicitement du KDD. Par la suite plusieurs tentatives de redéfinition ont été proposées comme par exemple :"Data mining, also referred to as knowledge discovery from data (KDD), is ́the automated or convenient extraction of patterns representing knowledge implicitly stored or captured in large databases, datawarehouses, the Web other massive information repositories, or data streams" [47]. Mais, aucune de ces définitions ne s'est réellement imposée.

Il faut noter, à la lecture des différents documents qui traitent du KDD, on peut se dire que, finalement, cela fait plusieurs années qu'on le pratique avec ce qu'on appelle l'analyse

Cust	*Cid*	*Cname*	*Caddr*
	c_1	John	Paris
	c_2	Mary	Paris
	c_3	Jane	Paris
	c_4	Anne	Tours

Prod	*Pid*	*Ptype*
	p_1	milk
	p_2	beer
	p_3	milk

Sales	*Cid*	*Pid*	*Qty*
	c_1	p_1	10
	c_2	p_3	3
	c_4	p_1	10
	c_3	p_3	5
	c_4	p_2	5
	c_4	p_3	3

$\mathcal{F}$: $Cid \rightarrow CnameCaddr$
$Pid \rightarrow Ptype$
$CidPid \rightarrow Qty$

$\mathcal{I}$: $\pi_{Cid}(Sales) \subseteq \pi_{Cid}(Cust)$
$\pi_{Pid}(Sales) \subseteq \pi_{Pid}(Prod)$

FIGURE 1.1 – Base de données introductive

de données et les techniques exploratoires.
En réalité, ce n'est pas aussi simple car la Fouille de données possède certaines particularités car elle utilise plusieurs disciplines comme :

- l'intelligence artificielle,
- les statistiques,
- et les bases de données.

En effet, avec le développement technologique, les données ne sont plus dans des tables transactionnelles mais dans des bases de données relationnelles construites sciemment pour une exploitation à des fins d'analyse, ce sont ces bases de données que l'on appelle entrepôts de données ou datawarehouse (en anglais).
Malheureusement, la plupart des techniques proposées sont dédiées au problème classique de la recherche d'itemsets fréquents (ou problème du panier de la ménagère) où les données résident dans des table transactionnelles.

C'est ainsi, que la communauté scientifique a commencé à parler de découverte de connaissances dans les bases de données relationnelles ou dans les entrepôts de données. Mais, l'extraction de ces informations est très difficile et pose un certain nombre de difficultés :

- Le temps d'exécution du fait de la taille importante et de la complexité des données à traiter ;
- L'espace de recherche est exponentiel en fonction de la taille des données ;
- Il n'existe pas un langage de l'extraction de requêtes pertinentes unifié comme c'est le cas de SQL dans le cadre d'interrogation des bases de données.

Aujourd'hui, avec le développement des entrepôts de données construits avec des bases de données relationnelles dans les entreprises, il est crucial de trouver de nouvelles techniques pour extraire les motifs dans les bases de données relationnelles.

Cette thèse, s'intéresse au problème de la recherche de requêtes fréquentes dans les bases de données relationnelles. En effet, les requêtes fréquentes extraites permettent la découverte de connaissances intéressantes.

L'exemple suivant emprunté à [99] donne l'importance de la découverte de tels motifs, considérons la base de données des films sur internet imdb [1] qui contient tout type d'informations sur les films et les deux requêtes suivantes :

- Q_1 : acteurs ayant joué dans un film de type 'dramatique', et
- Q_2 : acteurs ayant joué dans un film de type 'dramatique' mais aussi dans des films de type 'comedie'

Si les réponses à Q_1 et Q_2 contiennent 1000 et 900 tuples : nous pouvons alors déduire que des acteurs ayant joué dans un film 'dramatique' ont également joué dans un film 'comedie' qui est une régle d'association intéressante avec une certitude de 90%.

Bien sûr, cette connaissance peut être obtenue en utilisant les techniques classiques de découvertes d'itemsets fréquents. Mais l'application de ces techniques requière la transformation de la base de données en créant pour chaque acteur une transaction contenant la liste des genres de films dans lesquels il a joué.
De même, considérons le motif 66% des films joués par Jean Paul Belmondo ont également Alain Delon comme acteur. Ce motif peut être obtenu en considérant deux requêtes, une retournant les films joués par Jean Paul Belmondo et une autre retournant les films joués par Jean Paul Belmondo et Alain Delon simultanément. Ce motif peut être également obtenu par transformation de la base de données en une autre base de données transactionnelles différentes de la première, en créant pour chaque film une transaction contenant la liste de ses acteurs.
Il faut remarquer cependant qu'il est impossible d'avoir une transformation qui marche pour tous les motifs susceptibles de nous intéresser. En effet, nous avons compté par rapport au genre pour le premier cas et les acteurs pour le second. Ainsi, pour chaque type de motifs, il nous faut transformer la base de données, ce qui n'est pas réaliste.
De plus, il existe des motifs qui ne peuvent être obtenus en utilisant les techniques classiques comme par exemple le motif suivant : "80% des producteurs de films qui ont déjà été acteur dans un film donné ont joué au moins dans un film qu'ils ont produit", car dans ce cas précis on s'intéresse aussi bien aux acteurs qu'aux films, donc à deux schémas de projection différents. En effet, comme nous le verrons dans la suite la plupart des approches existantes ne s'intéressent qu'au cas où l'objet de comptage est fixe.
Cependant, ce motif peut être obtenu facilement grâce à deux requêtes, la première requête est celle demandant les producteurs-acteurs et la seconde celle demandant la liste des producteurs qui ont joué dans un film qu'ils ont produit.
Ainsi, la découverte de requêtes fréquentes permet également de découvrir d'autres types de motifs mettant en jeu des projections sur des schémas différents comme l'illustre l'exemple suivant :

Exemple 1.1. *Dans cette exemple, nous noterons par sup(Q) la cardinalité de la réponse à une requête Q donnée, considérons la base de données de la Figure 1.1 et les deux requêtes suivantes :*

- $Q_1 = \pi_{Cid}\sigma_{Caddr=paris}(Cust \bowtie Prod \bowtie Sales)$ *et*
- $Q_2 = \pi_{Cid}\sigma_{Caddr=paris,Ptype=lait}(Cust \bowtie Prod \bowtie Sales)$

Tout d'abord, il faut noter que la requête Q_1 est plus générale que la requête Q_2, en effet la requête Q_2 renvoie la liste des parisiens qui achètent du lait et la requête Q_1 les habitants

1. http ://www.imdb.com

parisiens. Comme $sup(Q_1) = sup(Q_2)$ *d'après l'exemple de la Figure 1.1, ceci veut dire tout parisien qui est dans la réponse à la requête* Q_2 *est également dans la réponse à la requête* Q_1*, d'où la règle d'association : "tous les clients qui habitent paris achètent du lait".*
De même en considérant les deux requêtes

- $Q'_1 = \pi_{Cid}\sigma_{Caddr=paris}(Cust \bowtie Prod \bowtie Sales)$ *et*
- $Q'_2 = \pi_{Cid}\sigma_{Caddr=paris,Ptype=lait}(Cust \bowtie Prod \bowtie Sales)$

comme $sup(Q'_1)/sup(Q'_2) = 2/3$ *nous pouvons alors déduire la règle d'association approximative vraie à 66% suivante : "les clients qui achètent du lait habitent paris", grâce au même raisonnement que précédemment.*
Considérons maintenant les deux requêtes suivantes :

- $Q_2 = \pi_{CnameCaddr}(Cust)$ *et*
- $Q_3 = \pi_{Caddr}(Cust)$

Remarquons tout d'abord que la requête Q_2 *renvoie plus ou autant de tuples que* Q_3 *car la projection de* Q_2 *est sur* $CnameCaddr$ *et celle de* Q_3 *sur* $Caddr$*, supposons qu'il existe deux valeurs distinctes de* $Caddr$ *pour un* $Cname$ *donné dans ce cas,* $sup(Q_2) > sup(Q_3)$*, comme* $sup(Q_2) = sup(Q_3)$*, d'après la base de données de la Figure 1.1, il n'existe pas deux valeurs distinctes de* $Cname$ *pour une même valeur* $Caddr$*, d'où la dépendance fonctionnelle* $C_{name} \rightarrow C_{addr}$*.*
De la même manière en considérant les deux requêtes :

- $Q'_2 = \pi_{Cid}(Sales)$ *et*
- $Q'_3 = \pi_{CidQ_{ty}}(Sales)$,

comme $sup(Q'_2)/sup(Q'_3) = 4/6 = 2/3$*, nous pouvons déduire que la dépendance fonctionnelle* $Cid \rightarrow Qty$ *est vérifiée à 66% en raisonnant de la même manière que précédemment.*
Nous rappelons qu'une dépendance fonctionnelle conditionnelle est une dépendance fonctionnelle vraie pour une condition de sélection donnée, considérons maintenant les deux requêtes suivantes :

- $Q_4 = \pi_{Cid}\sigma_{Caddr=paris}(Cust \bowtie Prod \bowtie Sales)$ *;*
- $Q_5 = \pi_{CidP_{type}}\sigma_{Caddr=paris}(Cust \bowtie Prod \bowtie Sales)$

comme $sup(Q_4) = sup(Q_5)$ *nous pouvons déduire la dépendance fonctionnelle conditionnelle suivante "les clients qui habitent paris achètent toujours le même type produit" avec la même remarque que pour les dépendances fonctionnelles.*
Finalement, en considérant les mêmes requêtes et en ajoutant le tuple : $c_3, p_1, 5$ *dans la table* $Sales$*,* $sup(Q_4) = 3$ *et* $sup(Q_5) = 4$*, la dépendance fonctionnelle conditionnelle précédente devient approximative et est vraie à 75%.*

Comme on vient de le voir dans l'exemple précédent, la recherche de requêtes fréquentes permet de découvrir des motifs très intéressants comme les régles d'association mettant en jeu des schémas différents par exemple le lien entre les producteurs et les films dans lesquels ils ont joué, les dépendances fonctionnelles, les dépendances fonctionnelles approximatives, les dépendances fonctionnelles conditionnelles, les dépendances fonctionnelles conditionnelles qu'il est impossible d'obtenir avec les techniques classiques de recherche d'itemsets fréquents, même moyennant une transformation. Cependant, il n'existe que quelques approches qui s'intéressent au problème de la recherche de requêtes dans les bases de données relationnelles sans transformation de celles-ci. De plus, la plupart d'entre elles considèrent

un schéma fixe pour le comptage ([30, 38, 59, 92]) comme pour le problème de la recherche d'itemsets fréquents. En effet, les requêtes considérées dans ces approches sont des projections sur un schéma fixe. Il est évident que ces approches ne peuvent pas nous permettre de découvrir des motifs mettant en jeu des schémas différents.
Ceci est une grande limitation, dans la mesure où des motifs intéressants comme, sans être exhaustif, des dépendances fonctionnelles, des dépendances fonctionnelles conditionnelles, des dépendances fonctionnelles approximatives, vues plus haut, ne peuvent pas être découvertes par ce type d'approche.
A notre connaissance, les seules approches qui s'intéressent aux problèmes de la recherche des requêtes fréquentes dans le cas où l'ensemble des schémas de projection est variable sont `Warmer` ([54]) et `Conqueror` ([56]).
Cependant, ces deux approches ne prennent pas en compte les dépendances fonctionnelles. De plus, `Warmer` génère des requêtes redondantes et la seconde, `Conqueror` ([56]) ne prend pas en compte les dépendances fonctionnelles et les dépendances d'inclusion.
En effet, l'approche `Warmer` génère beaucoup de requêtes redondantes et leur élimination pose un certain nombre de problèmes d'efficacité sachant que le test d'équivalence de deux requêtes est *NP-complet* ([71]).
La non prise en compte des dépendances fonctionnelles et des dépendances d'inclusion implique la génération de requêtes équivalentes relativement à celles-ci, ce qui crée des problèmes d'efficacité. En effet, deux requêtes équivalentes relativement aux dépendances fonctionnelles et dépendances d'inclusion ont même support, ce qui implique qu'un seul calcul de support par classe d'équivalence permettrait d'évaluer le support de toutes les requêtes appartenant à une classe d'équivalence donnée.
De plus, il est évident que la prise en compte de l'ensemble des schémas de projection possibles crée un certain nombre de difficultés car l'ensemble des schémas de projection possibles est exponentiel en fonction du nombre d'attributs de la base de données relationnelle, ce qui nécessite plusieurs optimisations.

La principale contribution de cette thèse est de proposer une approche permettant de générer efficacement sans redondances les requêtes candidates, mais également qui permet une évaluation efficace des supports des requêtes candidates.
Notre approche consiste à utiliser les propriétés structurelles des bases de données relationnelles. En effet, nous exploitons les dépendances fonctionnelles et les dépendances d'inclusion pour définir une relation de pré-ordre entre les requêtes. Cette relation de pré-ordre nous permettra de générer les requêtes candidates sans redondances, mais également de partitionner l'espace de recherche en classes d'équivalence, ce qui permettra d'évaluer une requête par classe d'équivalence (car comme nous allons le montrer deux requêtes appartenant à une même classe d'équivalence ont même support). De plus, comme nous le verrons, l'anti-monotonie de cette relation de pré-ordre nous permet d'utiliser un algorithme par niveau de type Apriori [8].
L'Exemple 1.2 donne une idée de notre approche :

Exemple 1.2. *Considérons la base de données représentée par la Figure 1.1, il est facile de vérifier que* $|\pi_{Cid}(Cust)| \geq |\pi_{Caddr}(Cust)|$ *car la dépendance fonctionnelle* $C_{id} \rightarrow C_{addr}$ *est vérifiée dans la table* $Cust$.

De même, il est facile de vérifier que nous avons $|\pi_{Cid}(Cust)| \geq |\pi_{Cid}(Sales)|$, car *la dépendance d'inclusion* $\pi_{Cid}(Sales) \subseteq \pi_{Cid}(Cust)$ *est vérifiée dans* Δ.
Egalement, nous pouvons vérifier que $|\pi_{Cid}(Cust)| \geq |\pi_{Cid}(Sales)|$, car $\pi_{Cid}(Sales) \subseteq \pi_{Cid}(Cust)$ *dans* Δ.
Nous pouvons également vérifier que $|\pi_{Cid}(\sigma_{Ptype=\texttt{beer}}(Prod \bowtie Sales))| \geq |\pi_{Cid\,Qty}(\sigma_{Pid=\texttt{p}_2}(Prod \bowtie Sales))|$.
Nous avons cette inégalité car *les dépendances fonctionnelles* $Cid\,Pid \rightarrow Cid\,Qty$ *et* $Pid \rightarrow Ptype$ *sont dans* $\mathcal{F}$, *et, dans* $Prod$, $\texttt{p}_2$ *est associé à* `beer`.
De plus, nous pouvons vérifier que $|\pi_{Cid}(\sigma_{Ptype=\texttt{beer}}(Prod \bowtie Sales))| \geq |\pi_{Cid\,Qty}(\sigma_{Pid=\texttt{p}_2}(Sales))|$.
Cette égalité est une conséquence de la précédente, car *le fait que* $\pi_{Pid}(Sales) \subseteq \pi_{Pid}(Prod)$ *soit dans* Δ, *implique dans* Δ *que :* $|\pi_{Cid\,Qty}(\sigma_{Pid=\texttt{p}_2}(Sales))| = |\pi_{Cid\,Qty}(\sigma_{Pid=\texttt{p}_2}(Prod \bowtie Sales))|$.

Dans le chapitre contribution, nous allons formaliser notre approche.
Cette thèse est organisée en cinq chapitres.
Le deuxième chapitre fait un état de l'art de l'extraction de motifs intéressants.
Le troisième chapitre est la contribution de cette thèse à l'extraction de connaissances dans les bases de données relationnelles organisées en schéma étoile. Il sera divisé en trois parties, la première partie présente les notations, concepts et formalisme utilisés, la deuxième partie présente comment les dépendances fonctionnelles et les dépendances d'inclusion peuvent réduire l'espace de recherche, la troisième partie présente notre algorithme de recherche de requêtes fréquentes, et la dernière partie traite de la complexité de notre algorithme.
Le quatrième chapitre présente l'implémentation des différents algorithmes proposés dans la partie contribution ainsi que leur expérimentation.
Enfin, le chapitre 5 sera la conclusion, dans laquelle nous résumons les principales contributions proposées dans le mémoire, et présentons un certain nombre de directions de recherches pouvant être envisagées à partir de nos travaux.

2
ETAT DE L'ART

2.1 Introduction

Dans cet état de l'art, nous allons tout d'abord tenter de définir le concept de **Fouilles de Données** ou **Data Mining**, puis nous allons présenter le formalisme générique de recherche de motifs proposé par Mannila et Toivonen. Ensuite, nous verrons son application à la recherche d'itemsets fréquents dans les bases de données transactionnelles, puis, nous verrons comment ce concept s'applique à la recherche de motifs fréquents dans le cas des bases de données relationnelles.

Finalement, la recherche de requêtes fréquentes étant le sujet de cette thèse, nous présenterons quelques méthodes existantes permettant la recherche de tels motifs.

2.2 Le Datamining ou Fouille de données

Le Datamining fait référence à l'extraction de motifs potentiellement intéressants dans de grandes masses de données. Il faut noter que le datamining n'est qu'une partie d'un processus global comme l'illustre la Figure 2.1.

Dans ce processus, la phase de préparation (sélection et transformation) consiste à sélectionner la partie des données de la base globale qui nous intéresse. La phase de préparation des données (pré-processing) consiste quant à elle, à nettoyer et à intégrer les données. Les données obtenues sont éventuellement transformées sous un format donné pour l'étape suivante : c'est la phase de transformation, les données obtenues après la phase de transformation, comme le montre la Figure 2.1, seront les données d'entrées de la phase de Data Mining. La phase de Data Mining recherche et retourne les connaissances découvertes. Ces connaissances seront ensuite évaluées, interprétées et présentées à l'utilisateur pour une exploitation aisée : c'est la phase d'interprétation et d'évaluation.

Pour donner un cadre formel au problème d'extraction de motifs fréquents, un formalisme générique a été proposé par [89]. Dans la section suivante, nous allons décrire ce formalisme.

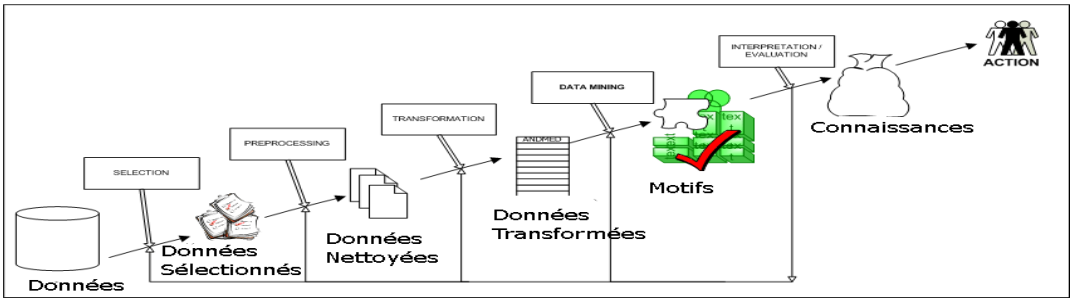


FIGURE 2.1 – Le Data Mining : une partie du processus global d'extraction de motifs intéressants.

2.3 Recherche de motifs fréquents

2.3.1 Formalisme général de recherche de motifs intéressants

Mannila et Toivonen ont défini un formalisme général pour la découverte de motifs fréquents à partir d'un ensemble de données [89]. Ce formalisme se définit comme suit : soit S un ensemble de données, $\mathcal{L}$ un langage d'expression de propriétés sur S appelées motifs, et un critère de qualité q tel que pour tout motif $\varphi \in \mathcal{L}$ par : $q(\varphi, S) = true$ si et seulement si φ est intéressant dans S.
La recherche de motifs intéressants peut être vue comme la construction d'une théorie $\mathcal{Th}(\varphi, S, q)$ telle que $\mathcal{Th}(\varphi, S, q) = \{\varphi \in \mathcal{L} \mid q(\varphi, S) = true\}$.
Sous certaines conditions, cette théorie peut être construite avec un algorithme par niveau que nous présenterons par la suite. Auparavant, nous allons donner quelques définitions des concepts utilisés par ce formalisme.

Définition 2.1. *Une relation d'ordre partiel sur les motifs de $\mathcal{L}$ est appelée relation de spécialisation sur $\mathcal{L}$. Cette relation de spécialisation est notée $\preceq$.*
Soient φ, θ deux motifs de $\mathcal{L}$, on dit que le motif φ est plus générale que θ si et seulement si $\varphi \preceq \theta$. On dit également que le motif θ est plus général que φ.

Définition 2.2. *Un ensemble de motifs $\mathcal{L}$, muni de la relation de spécialisation $\vee$, est un **inf-semi-treillis**, si chaque couple de motifs (φ_1, φ_2) admet un plus petit majorant commun unique, noté $\varphi_1 \vee \varphi_2$ qui est le motif le plus général qui soit moins générale que φ_1 et φ_2, i.e. $\varphi_1 \vee \varphi_2 = min_{\preceq}\{\varphi \in \mathcal{L} \mid \varphi_1 \preceq \varphi, \varphi_2 \preceq \varphi\}$.*
*Un ensemble de motifs $\mathcal{L}$, muni de la relation de spécialisation est un **treillis**, si chaque couple de motifs φ_1, φ_2 admet un plus petit majorant commun unique, $\varphi_1 \vee \varphi_2$ et un plus grand minorant commun, noté $\varphi_1 \wedge \varphi_2$ qui est le motif le moins générale qui soit plus général que φ_1 et φ_2, c'est à dire, $\varphi_1 \wedge \varphi_2 = max_{\preceq}\{\varphi \in \mathcal{L} \mid \varphi \preceq \varphi_1, \varphi \preceq \varphi_2\}$.*

Définition 2.3. *Un opérateur de généralisation ρ sur un espace de recherche $\mathcal{L}$ est une application de $\mathcal{L}$ dans $2^{\mathcal{L}}$ telle que :*
– $\forall(\varphi, \varphi') \in \mathcal{L}^2 : \varphi' \in \rho(\varphi)$ *si* $(\varphi' \prec \varphi)$ *et il n'existe pas de* $\varphi'' \in \mathcal{L}$ *tel que* $\varphi' \prec \varphi'' \prec \varphi$.

Pour tout $X \subseteq \mathcal{L}$ *on note :*$\gamma(X) = \bigcup_{\varphi \in X} \rho(\varphi)$,$\gamma^*(X) = \bigcup_{i=0}^{+\infty} \gamma^i(X)$.
$\gamma^*(X)$ *regroupe l'ensemble des motifs obtenus par applications successives de* ρ *sur les différents motifs de* X *et leurs spécialisations.On l'appelle la* ***clôture transitive*** *de* X.

Définition 2.4. *Un opérateur de généralisation* ρ *sur un ensemble de recherche* $\mathcal{L}$ *est dit* ***complet*** *relativement à* $L \subseteq \mathcal{L}$ *si* $\gamma^*(X) = \mathcal{L}$,*où* $X = \{\varphi \in L | (\nexists \varphi' \in L)(\varphi \prec \varphi')\}$.

Une relation de spécialisation sur $\mathcal{L}$ induit souvent une structure de treillis et un opérateur de spécialisation ρ sur $\mathcal{L}$. L'algorithme générique de Mannila et Toivonen utilise ainsi cette relation de spécialisation pour le parcours du treillis.

Définition 2.5 (Monotonie du critère de qualité)**.** *Soit* q *un critère de qualité,* q *est dit* ***monotone*** *si* $\forall(\varphi, \gamma) \in \mathcal{L}^2$, *si* $\varphi \preceq \gamma$ *et* $q(\gamma, S) = true$ *alors* $q(\varphi, S) = true$.

L'algorithme générique de construction de $\mathcal{TH}(\varphi, S, q)$ fonctionne avec un critère de qualité anti-monotone, dont la propriété peut être reprise comme suit : si un motif φ est intéressant, alors tous les motifs plus généraux que lui le sont aussi.
Ceci est l'idée de base de la proposition suivante [89, 93, 76] :

Proposition 2.1. *Si* $(\mathcal{L}, \preceq)$ *est un* ***treillis*** *et le critère de qualité* $\boldsymbol{q}$ *est anti-monotone par rapport à* $\preceq$, *alors* $\mathcal{TH}(\varphi, S, q)$ *est un* ***inf-semi-treillis***.

2.3.2 Algorithme général de recherche de motifs intéressants

Algorithm 1 Algorithme Générique de Mannila et Toivonen

Entrées: Un ensemble de données S, un langage $\mathcal{L}$ muni d'une relation de spécialisation $\preceq$ et un prédicat de sélection q monotone.
Sortie: L'ensemble $\mathcal{TH}(\varphi, S, q)$.
1: $C_1 = \{\varphi \in \mathcal{L} : \nexists \varphi', \varphi' \preceq \varphi\}$;
2: i=1 ;
3: **Tant Que** $(C_i \neq \emptyset)$ **do**
4: //**Phase d'évaluation**
5: $F_i = \{\varphi \in C_i : q(\varphi, S) = true\}$;
6: //**Phase de génération**
7: $C_{i+1} = \{\varphi' \in \rho(\varphi) | \varphi \in F_i\}$;
8: $C_{i+1} = \{\varphi' \in C_{i+1} | (\forall \varphi \in \mathcal{L})(\varphi' \in \rho(\varphi) \Rightarrow \varphi \in F_i)\}$;
9: $i = i + 1$
10: **Fin Tant Que**
11: **Retourner** $\bigcup_{i<j} F_i$

L'Algorithme 1 parcourt niveau par niveau le treillis $(\mathcal{L}, \preceq)$ et construit les motifs intéressants en partant des motifs les plus généraux, en alternant :

– une phase d'évaluation des motifs candidats de niveau i. Durant cette phase, seuls les candidats qui vérifient le critère de qualité sont retenus (cf. ligne 5).

– une phase de génération des motifs candidats de niveau $i+1$ à partir des motifs intéressants de niveau i.
Cette phase se décompose en deux : dans un premier temps, les motifs intéressants de niveau i sont spécialisés (cf. ligne 7), ensuite, ne sont retenus que les motifs dont tous les sous-motifs sont intéressants (cf. ligne 8). Ces deux phases sont respectivement appelées phases de spécialisation et d'élagage. Nous verrons dans la suite que le formalisme de Mannila s'applique au problème de la recherche d'itemsets fréquentes dans les bases de données transactionnelles, mais aussi à la découverte de motifs dans les bases de données relationnelles.

2.4 Recherche d'itemsets et règles fréquentes

Un exemple typique d'extraction de motifs dans une base de données est la recherche d'itemsets fréquents dans les bases de données transactionnelles.

Actuellement, le problème de la recherche d'itemsets a inspiré beaucoup d'autres applications mettant en oeuvre d'autres types de motifs (itemsets séquentiels, arbres, graphes, requêtes, etc...).

Ce problème a fait l'objet de plusieurs travaux qui prennent principalement leur source des travaux de Agrawal et al. [8]. Cependant, nous montrons dans cette section comment ce problème s'exprime en utilisant le formalisme de Mannila et Toivonen [89].
Soit $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$ un ensemble d'items. Un sous-ensemble $I = \{i_1, i_2, \ldots, i_k\} \subseteq \mathcal{I}$ est appellé *itemset* ou *k-itemset* lorsqu'il contient k éléments.
Soit $\mathcal{D}$ la base de données de *transactions* notées T, dans laquelle chaque $T = (t_{id}, I_{t_{id}})$ est un couple tel que t_{id} est l' identifiant unique de la transaction et $I_{t_{id}}$ est un *itemset*.
On dit qu'une transaction $T = (t_{id}, I_{t_{id}})$ contient ou supporte un itemset I si et seulement si $I \subseteq I_{t_{id}}$.
Nous montrons que ce problème est une instance du formalisme générique de Mannila et Toivonen :

– L'ensemble S de données est représenté ici par la base de données de transactions $\mathcal{D}$.
– Les motifs $\mathcal{L}$ sont ici les itemsets.
– Le critère de qualité q d'un itemset est son support, c'est à dire le nombre de transactions le contenant.
– La relation de spécialisation ($\preceq$) est ici l'inclusion ensembliste ($\subseteq$).
– L'opérateur de spécialisation est ici la relation d'inclusion stricte ($\subset$).

Il est à noter également que le critère de qualité (support) des itemsets est anti-monotone relativement à ($\subseteq$) et est défini pour tout motif $\varphi \in \mathcal{L}$ par : $q(\varphi, S) = true$ si et seulement si le support de φ est supérieur à un seuil donné que nous notons *minsup*.

Nous pouvons alors utiliser l'Algorithme 1 proposé par Mannila et Toivonen pour trouver les itemsets fréquents. Cependant, nous ne pouvons pas parler de recherche d'itemsets fréquents dans les bases de données transactionnelles sans parler de l'algorithme Apriori proposé dans [8] qui est un algorithme très efficace. L'algorithme Apriori est une instance de l'algorithme générique proposé par Mannila et Toivonen, qui calcule tous les item-

sets fréquents et à partir d'eux trouve les règles d'association intéressantes en utilisant l'instanciation précédente. Cette algorithme est décrit par l'Algorithme 2.

Algorithm 2 Algorithme Apriori de Génération des itemsets fréquents

Entrées: Un ensemble de données $\mathcal{D}$, un seuil de support *minsup*.
Sortie: L'ensemble des itemsets fréquents $\mathbb{F}\mathtt{req}_{minsup}(\mathcal{D})$.

1: $C_1 = \{\{i\} | i \in \mathcal{I}\}$
2: k=1 ;
3: **Tant Que** $C_k \neq \{\}$ **do**
4: //**Phase d'évaluation**
5: **Pour Tout** transactions $(t_{id}, I) \in \mathcal{D}$ **do**
6: **Pour Tout** candidats itemsets $X \in C_k$ **do**
7: **Si** $X \subseteq I$ **Alors**
8: $sup(X) + +$
9: **Fin Si**
10: **Fin Pour Tout**
11: //**Extraire les itemsets fréquents**
12: $\mathcal{F}_k \leftarrow \{X | sup(X) \geq minsup\}$
13: **Fin Pour Tout**
14: **Pour Tout** $X, Y \in \mathcal{F}_k, X[i] = Y[i]$ pour $1 \leq i \leq k-1$, et $(X[k] < Y[k])$ **do**
15: $I \leftarrow X \cup \{Y[k]\}$
16: **Si** $\forall J \subset I, |J| = k : J \in \mathcal{F}_k$ **Alors**
17: $C_{k+1} \leftarrow C_{k+1} \cup J$
18: **Fin Si**
19: **Fin Pour Tout**
20: **Fin Tant Que**
21: **Retourner** $\bigcup_{i<k} F_i$

2.5 Recherche de motifs dans les bases de données relationnelles

Une base de données relationnelle est une base de données contenant plusieurs tables relationnelles.

Egalement, nous pouvons avoir plusieurs types de motifs dans une base de données relationnelle : les itemsets relationnels fréquents qui sont une généralisation du problème de la recherche d'itemsets dans une table transactionnelle au cas relationnel, mais également d'autres motifs comme les requêtes fréquentes qui feront l'objet de notre contribution dans cette thèse.

Cependant, à cause de la structure particulière des bases de données relationnelles, il apparait que les techniques classiques de recherche de motifs dans les bases de données transactionnelles s'applique difficilement au cas relationnel. En effet, les bases de données relationnelles contiennent plusieurs tables, chaque table a un ensemble d'attributs

TID	$Itemsets$
1	{bière,couches}
2	{bière,couches,lait}
3	{lait}
4	{bière,couches,chocolat}
5	{bière,couches,lait}
6	{couches,lait}

FIGURE 2.2 – Exemple d'une table de transactions (1)

t_1	a,b
t_2	c,d,e
t_3	a,b,c

FIGURE 2.3 – Exemple d'une table transactionnelle (2)

et chaque attribut a une signification particulière contrairement aux tables transactionnelles où l'ensemble des items ont tous la même signification (car appartenant au même domaine).
La recherche d'itemsets fréquents dans une table transactionnelle dont le schéma est l'ensemble des items possibles, peut être vue comme un cas particulier des itemsets dans une table relationnelle dans laquelle chaque tuple a au plus deux valeurs '0' et '1' sur un attribut donné, par exemple la table transactionnelle 2.3 peut être remplacée par la table relationnelle 2.4.

Une table transactionnelle est alors une table relationnelle binaire dans laquelle seules deux valeurs sont permises.
Dans le cas général, une table relationnelle peut avoir plus de 2 valeurs différentes pour chaque attribut de la table, en effet, les domaines d'attributs d'une base de données ont souvent plus de deux valeurs différentes. Ceci implique que les méthodes classiques dédiées aux bases de données transactionnelles ne peuvent pas être appliquées directement aux bases de données relationnelles. Cependant, une solution possible pour appliquer ces techniques serait de considérer les paires *attribut-valeur* comme étant des items [7]. Ceci se fait, en transformant une base de donnée relationnelle en une table transactionnelle, par exemple la table relationnelle représentée par la Figure 2.5 est transformée en la table transactionnelle représentée par la Figure 2.6 avec, P_i représentant la paire $(Produit = i)$,

a	b	c	d	e
1	1	0	0	0
0	0	1	1	1
1	1	1	0	0

FIGURE 2.4 – Table transactionnelle sous forme d'une table relationnelle

M_i représentant la paire $(Magasin = i)$, et A_i représentant la paire $(Adresse = i)$. Cet exemple montre que l'on obtient ainsi une table similaire au cas transactionnel, où les items sont remplacés par des paires de valeurs.

Il apparait cependant que pour certains types de base de données relationnelles, cette transformation n'est pas toujours réaliste, et même dans les cas où cette transformation est possible, beaucoup d'informations utiles sont perdues. En effet, chaque attribut d'une table relationnelle renferme une sémantique particulière, également, les dépendances fonctionnelles et les dépendances d'inclusion entre les données sont perdues.

OID	$Produit$	$Magasin$	$Adresse$
1	Bière	Auchan	Defense
2	Vin	Auchan	Paris
3	Lait	Leclerc	Paris
4	Jus d'Orange	Leclerc	Defense
5	Vin	Auchan	Paris

FIGURE 2.5 – Représentation Classique d'une table relationnelle

1	P_{Biere}	M_{Auchan}	$A_{Defense}$
2	P_{Vin}	M_{Auchan}	A_{Paris}
3	P_{Lait}	$M_{Leclerc}$	A_{Paris}
4	$P_{Jusd'Orange}$	$M_{Leclerc}$	$A_{Defense}$
5	P_{Vin}	M_{Auchan}	A_{Paris}

FIGURE 2.6 – Représentation d'une table relationnelle sous forme Transactionnelle

Comme énoncé précédemment, durant ces dernières décennies, le problème de la découverte de motifs intéressants dans les bases de données contenant une seule table a été bien étudié et des approches efficaces ont été proposées. Cependant, les approches classiques de recherche de motifs intéressants ont de réelles limites quand il s'agit d'une base de données relationnelle et ceci pour plusieurs raisons :

- La pauvreté du langage d'expression utilisé : les motifs sont exprimés dans un formalisme pauvre (attribut-valeur) qui a la puissance d'expression de la logique propositionnelle or comme nous le verrons dans la suite les types de motifs apparaissant dans les bases de données relationnelles s'expriment non pas avec la logique propositionnelle mais plutôt, avec la logique du premier ordre.
- Les approches classiques ne portent que sur des données stockées dans une seule table.

Ainsi, des approches prenant en compte les spécificités du modèle multidimensionnel ou multi-relationnel ont été proposées [45, 43, 42, 31, 10, 13, 56] et elles constituent un domaine spécifique appelé ***MRDM*** (Multi Relational Data Mining).
Ces approches peuvent être classées en deux catégories :

– La recherche se fait en utilisant la programmation logique inductive (PLI) qui se base sur la logique du premier ordre pour représenter les données et les motifs et sur des biais syntaxiques et sémantiques pour réduire l'espace de recherche.
– La recherche se fait en considérant tous les motifs de la base, cette approche est celle utilisée dans cette thèse, nous y reviendrons plus amplement dans notre chapitre contribution.

Dans ce qui suit, nous allons présenter quelques concepts liées à l'extraction de motifs dans le cas des bases de données relationnelles, puis nous présenterons dans les section suivantes quelques approches de découvertes de motifs dans les bases de données relationnelles.

2.5.1 Formalisme de la découverte de motifs dans les bases de données relationnelles

Dans cette section, nous allons d'abord étudier quelques définitions de base de la logique du premier ordre pour comprendre comment les données sont représentées. Le lecteur désirant une présentation plus complète, pourra consulter le livre [84].
Le langage logique du premier ordre est défini par : un ensemble de symboles, de variables $\{v1...v_n\}$, un ensemble de symboles de constantes $\{c_1...c_m\}$, un ensemble de symboles de fonctions $\{f_1...f_k\}$, un ensemble de symboles de prédicats $\{p_1...p_j\}$, des connecteurs logiques $(\vee, \wedge, \neg, \Rightarrow, \Leftrightarrow)$, des quantificateurs $(\exists, \forall)$, des symboles de ponctuation.
A chaque symbole de prédicat ou de fonction est associée une arité qui désigne le nombre d'arguments auxquels ce symbole doit s'appliquer.
Nous ne détaillons pas ici la construction des formules qui est exposée dans [84]. Nous rappelons toutefois que : si p est un symbole de prédicat d'arité n, et si $t_1...t_n$ sont des termes, alors $p(t_1...t_n)$ est appelé un atome.
Il faut de plus signaler qu'en général, les fonctions ne sont pas considérées dans le cadre de la recherche de motifs dans les bases de données relationnelles.
De plus, un fait est un atome ne comportant aucune variable. Un ensemble de données est un ensemble de faits.

Dans ce qui suit, nous parlerons invariablement d'ensemble de données ou de faits.

Définition 2.6 (***Domaine***). *Soit p un prédicat d'arité m et* $1 \leq k \leq m$. *On appelle dom(k,p) l'ensemble des valeurs prises par le k-ème terme d'un atome sur p. Ces ensembles de valeurs sont appelés* ***domaines****.*

Exemple 2.1. *Soit* ***Pred={Consommateur, Vente, Produit}*** *un ensemble de prédicats d'arités respectives 3,2,2 défini comme suit :*
– ***Consommateur****(*$C_{id}, C_{prof}, C_{adresse}$***)** *qui signifie : le consommateur* C_{id} *ayant pour profession* C_{prof} *habite à l'adresse* $C_{adresse}$.
– ***Vente****(*C_{id}, P_{id}***)** *qui signifie : le consommateur* C_{id} *a acheté le produit* P_{id}.
– ***Produit****(*P_{id}, P_{type}***)** *qui signifie : le produit* P_{id} *a pour libellé* P_{type}.

Nous pouvons alors avoir les domaines suivants :
dom(1,Consommateur)=dom(1,Vente)*={olivier,cheikh,françois,augustine,nicolas},*
dom(2,Consommateur)*={enseignant,avocat,plombier},*

dom(3,Consommateur)*={paris,dakar,ziguinchor},*
dom(2,Vente)=dom(1,produit)*={1,2,3,4},*
dom(2,Produit)*={lait,bière,jus d'orange,vin}.*

Voici un exemple d'ensemble de données :

S={Produit(1,lait),Produit(2,bière),Produit(3,jus d'orange),
Produit(4,vin),Consommateur(cheikh,enseignant,ziguinchor)
Consommateur(olivier,avocat,paris),Consommateur(nicolas,enseignant,paris),
Consommateur(augustine,plombier,dakar),Vente(cheikh,1),Vente(olivier,2),
Vente(nicolas,2)}

Cet exemple sera utilisé dans la suite comme exemple de référence.
Dans la littérature dédiée, il existe principalement deux types de motifs les **requêtes conjonctives** et les **implications entre requêtes conjonctives** qui sont respectivement les équivalents des itemsets et des règles d'association du contexte transactionnel.

Définition 2.7 (**Requêtes Conjonctives**). *Une requête conjonctive Q est une formule de la forme :* $\exists Y Q_1 \wedge Q_1 \wedge ...Q_p$ *avec* $Q_1 \wedge Q_2 \wedge ...Q_p$ *une conjonction d'atomes, Y un ensemble de variables apparaissant dans* $Q_1 \wedge Q_2 \wedge ...Q_p$*, les variables de* $Q_1 \wedge Q_2 \wedge ...Q_p$ *qui ne sont pas dans Y sont appelées* ***variables libres****.*

Pour pouvoir donner la définition de ce qu'est la réponse à une requête, donnons la définition d'une substitution [84].

Définition 2.8 (**Substitution**). *Une substitution est une application de l'ensemble des variables vers l'ensemble des constantes et des variables.*

Par exemple $\theta = \{x/cheikh, y/z\}$ est la substitution définie par : $\{\theta(x) = cheikh, \theta(y) = z\}$.
On note $p\theta$ le résultat de la substitution θ à l'ensemble des termes de l'atome p.
Une substitution θ est dite élémentaire si :

- Il existe une variable unique x_0 telle que $\theta(x_0)$ soit une constante et,
- Pour toute variable y différente de x_0, nous avons $\theta(y) = y$.

Définition 2.9 (**Réponse à une requête conjonctive**). *Soit Q une requête conjonctive. Pour tout ensemble de faits S, la réponse à Q dans S, notée Q(S) est l'ensemble des n-uplets* $K\theta$ *où :*

- *K est l'ensemble des variables libres de Q et,*
- θ *est une substitution telle que pour tout atome* $p(t_1...t_n)$ *de Q,* $p\theta$ *appartient à S.*

Remarque :
Quand on parle d'extraction le plus important est le sujet de l'extraction, c'est à dire ce que l'on compte, ou encore ce à quoi on s'intéresse.
Dans le cas transactionnel, il n'y a pas d'ambiguïté car nous avons des motifs de la forme : 50% des transactions contenant de la bière contiennent aussi des chips et des couches. Par contre dans le cas d'une base de données contenant plusieurs tables, il faut définir un

sujet d'extraction car à priori nous ne savons pas ce à quoi nous nous intéressons. Si nous prenons notre exemple précédent, on peut s'intéresser à n'importe quel attribut par exemple, aux professions, aux adresses, ou aux produits.
Il est donc nécessaire de définir l'objet de l'extraction, ceci se fait en définissant ***une requête de référence***.
Une requête de référence est une requête conjonctive qui précise le sujet de l'extraction.
Lorsque cette requête est constituée d'un seul atome, on parle d'***atome de référence***.
Par exemple, si on s'intéresse aux consommateurs on peut définir la requête de référence : $(\exists y, z)(Consommateur(x, y, z))$.
La recherche des motifs se présente ainsi comme suit : étant donné une requête de référence R de variables libres K, on s'intéresse aux requêtes conjonctives de la forme $R \wedge Q$ où Q est une requête de même ensemble de variables libres K que R.

Exemple 2.2. *Considérons la requête de référence* $R = (\exists y_1)(Vente(x_1, y_1))$*, la requête* $Q = (\exists x_2, y_1)(Consommateur(x_1, x_2, paris) \wedge Vente(x_1, y_1) \wedge Produit(y_1, biere))$ *s'intéresse alors aux consommateurs parisiens qui achètent de la bière.*

Définition 2.10 (**Implication entre requêtes**). *Soit R une requête de référence de variables libres K,* Q_B *et* Q_H *deux requêtes conjonctives de même ensemble de variables libres K.*
On s'intéresse alors aux implications logiques de la forme : $(\forall K)((R \wedge Q_B) \Rightarrow (R \wedge Q_H))$.

Remarque : Les implications sont les équivalents des règles d'associations dans le cas transactionnel.

Exemple 2.3. *Soit la requête de référence* $R = (\exists y_1)(Vente(x_1, y_1))$.
Soient $Q_1 = (\exists x_3, y_1)(Consommateur(x_1, Enseignant, x_3) \wedge Vente(x_1, y_1))$
et $Q_2 = (\exists y_1)(Vente(x_1, y_1) \wedge Produit(y_1, Biere))$*. La requête* Q_1 *a pour réponse les noms des consommateurs enseignants .*
La requête Q_2 *a pour réponse l'ensemble des consommateurs de bière.*
$(\forall x_1)(Q_1 \Rightarrow Q_2)$ *indique que tous les enseignants sont consommateurs de bière.*

Il est à noter, cependant que dans certaines publications comme [31], les motifs sont exprimés dans un formalisme ***DATALOG*** [24].
Nous allons maintenant présenter dans ce qui suit, les mesures d'intérêt des motifs et leurs propriétés. Les deux mesures les plus utilisées sont le support et la confiance.

Définition 2.11 (**Support d'une requête**). *Étant donnée une requête de référence R, soit Q une requête conjonctive de même ensemble de variables libres que R.*
Pour tout ensemble de faits S, le support de Q dans S relativement à R est défini par : $sup(Q/R, S) = |R \wedge Q(S)|/|R(S)|$.

Définition 2.12 (**Confiance d'une implication**). *Étant donnée une requête de référence R, soit* Q_B *et* Q_H *deux requêtes conjonctives de même ensemble de variables libres K.*
Soit I l'implication : $(\forall K)(R \wedge Q_B \Rightarrow R \wedge Q_H)$*. Pour tout ensemble de faits S, la confiance de I dans S relativement à R est définie par :*
$conf(I/R, S) = |R \wedge Q_B \wedge Q_H|/|R \wedge Q_B| = sup(Q_B \wedge Q_H/R, S)/sup(Q_B/R, S)$.

Exemple 2.4. *Considérons notre exemple précédent (Exemple 2.3), et soit* $R = (\exists y_1)(Vente(x_1, y_1))$. *La requête* $Q_1 = (\exists y_1, x_2, x_3)(Consommateur(x_1, x_2, Paris) \wedge Vente(x_1, y_1))$ *définit l'ensemble des consommateurs Parisiens.*
La requête $Q_2 = (\exists y_1)(Vente(x_1, y_1) \wedge Produit(y_1, Biere))$ *définit l'ensemble des buveurs de bière.*
Soit $I_1 : Q_1 \Rightarrow Q_2$ *qui représente l'assertion suivante : tous les habitants de Paris acheté de la bière.*
$sup(Q_1/R, S) = |Q_1(S)|/|R(S)| = 2/3$
$conf(I/R, S) = |Q_1 \wedge Q_2(S)|/|R \wedge Q_2(S)| = 2/2 = 1.$

Dans le contexte des bases de données relationnelles, nous avons des requêtes comme motifs à la place des itemsets, une relation de spécialisation possible est alors l'inclusion entre requêtes, définie de la manière suivante :

Définition 2.13 (**Inclusion de requêtes**). *Soit P et Q deux requêtes conjonctives de même ensemble de variables libres. On dit que P est incluse dans Q noté $P \subseteq Q$, si $P(S) \subseteq Q(S)$ pour tout ensemble de données S. De plus, si $P \subseteq Q$, alors l'implication $(\forall K)(P \Rightarrow Q)$ est vraie.*

Le support est anti-monotone par rapport à l'inclusion entre requêtes ce qui permet d'utiliser un algorithme par niveau de type Apriori, cependant il faut noter que le test d'inclusion de requêtes est un problème NP-complet ce qui rend difficile l'utilisation de l'inclusion comme relation de spécialisation entre requêtes.

2.5.2 Les critères de qualité

Soient ***minsup*** et ***minconf*** deux seuils respectivement de support et de confiance fixés par l'utilisateur.

Définition 2.14 (***Requête fréquente***). *Soit un ensemble de données S, une requête de référence R, et Q une requête conjonctive de même ensemble de variables libres que R. Q sera dite fréquente si et seulement si $sup(Q/R, S) \geq$* ***minsup***

Définition 2.15 (***Règle intéressante***). *Soit un ensemble de données S, une requête de référence R, Q_B et Q_H deux requêtes conjonctives de même ensemble de variables libres que R.*
L'implication I :$(\forall K)(R \wedge Q_B \Rightarrow R \wedge Q_H)$ est intéressante si et seulement si :
- $sup(Q_B \wedge Q_H/R, S) \geq$ ***minsup*** *et,*
- $conf(I/R, S) \geq$ ***minconf****.*

2.6 Algorithmes de recherches de motifs dans le cas multi-relationnel

La découverte de motifs dans les bases de données relationnelles pose un certain nombre de difficultés. Ces difficultés sont dues surtout à la taille de l'espace de recherche mais également la taille des motifs extraits, qui peut être très grande.

En effet, dans le cas transactionnel, l'espace de recherche est en $2^{|\mathcal{I}|}$ avec $\mathcal{I}$ représentant l'ensemble des itemsets. L'espace de recherche est encore plus grand dans le cas d'une base relationnelle car les données et les motifs s'expriment avec la logique du premier ordre.
Les approches classiques de découverte de motifs comme celui vu plus haut (cf. Section 2.4) ont alors de sérieuses limites quand il s'agit d'une base de données relationnelle.
C'est ainsi, que beaucoup d'algorithmes ont été proposés dans le but de restreindre ou d'explorer efficacement l'espace de recherche dans le cas d'une table relationnelle.
Les approches proposées peuvent être classées en deux catégories :

- Celles utilisant un biais syntaxique ou sémantique : les biais syntaxiques sont des restrictions sur la forme des motifs [45, 46].
  On peut citer, comme exemples de biais syntaxiques, ceux qui consistent à limiter par exemple le nombre de littéraux, ou à limiter le nombre de variables quantifiées existentiellement.
  Celles utilisant les biais sémantiques qui utilisent quant à eux des informations externes aux motifs [36, 32, 54].
  Ces informations, prennent généralement la forme d'un ensemble de contraintes comme le type des attributs figurant au niveau des arguments des prédicats qui constituent le motif [31].
- Les approches qui n'utilisent pas de biais mais qui utilisent une relation de spécialisation entre les requêtes pour réduire le parcours l'espace de recherche [56], la contribution de cette thèse que nous présenterons dans le chapitre suivant sera basée sur ce type approche.

Dans les sous-sections suivantes, nous allons présenter les approches les plus connues.

2.6.1 Approche Programmation Logique Inductive : `WARMR`

Nous allons présenter ici un algorithme appelé `WARMR` [31], qui utilise un formalisme appelé W_{rmode}. Le formalisme W_{rmode} qui prend ses sources de PROGOL [95], permet de déclarer le type des arguments mais aussi leurs modes.
Le biais syntaxique W_{rmode} se définit comme suit :

Définition 2.16 (***Contexte d'extraction***). *Un contexte d'extraction est un couple $C = \langle r, W \rangle$, où :*

- *r est un atome de référence.*
- *W est un ensemble d'atomes avec des contraintes de types et de modes.*
- *Chaque argument d'un atome de W peut être étiqueté par "-", "+", "-type", "+type", ou par "a".*

Lorsque l'argument est étiqueté par un "-", on parle de variable de sortie.
Lorsque l'argument est étiqueté d'un "+", on parle de variable en entrée.
Les arguments étiquetés par "-type" et "+type" sont ainsi appelés arguments typés. Les arguments étiquetés par "a" correspondent à des constantes.
L'étiquette permet de restreindre le choix des variables dans les arguments des prédicats. Cela se traduit de façon plus précise par l'application de contraintes de mode et de contraintes de type que les requêtes définies par ce contexte doivent respecter.

Définition 2.17 (***Contraintes de mode***). *Une requête vérifie les contraintes de mode s'il existe un ordre sur les atomes qui la constituent tel que pour chaque atome, chaque variable d'entrée ait au moins une occurrence parmi les atomes qui le précédent, et pour chaque atome, aucune variable de sortie n'ait une occurrence parmi les atomes qui le précédent.*

Définition 2.18 (***Contraintes de type***). *Une requête vérifie les contraintes de type si les arguments typés (des différents atomes) qui partagent un même nom de variable ont des types identiques.*

L'Exemple 2.5 suivant illustre le formalisme W_{rmode} défini plus haut.

Exemple 2.5. *Soit le contexte défini par* $C_1 = \langle p(x), \{p(+); q(+,-)\}\rangle$, *où* $p(x)$ *est l'atome de référence et* W *est l'ensemble* $p(+); q(+,-)$. *La requête* $(\exists z)(p(x) \wedge q(x,z)) \in Q(C_1)$, *alors que la requête* $(\exists z)(p(x) \wedge q(z,x) \notin Q(C_1)$, *car le premier argument de q (z) est une variable en entrée et devrait avoir une occurrence parmi les arguments des atomes qui le précédent, ce qui n'est pas le cas.*
Soient maintenant deux types différents s et t et le contexte défini par $C_2 = \langle p1(x), \{p1(-s); p2(+s,-t); p3(+t,a)\}\rangle$. $(\exists z)(p1(x) \wedge p2(x,z) \wedge p3(z,a)) \in Q(C_2)$, *mais* $(\exists z)(p1(x) \wedge p2(x,z) \wedge p3(x,a)) \notin Q(C_2)$ *car le deuxième argument de p2 et le premier de p3 doivent être de même type.*

Nous allons maintenant définir l'opérateur de spécialisation qui déterminera de façon plus formelle l'ensemble des requêtes qui appartiennent à l'espace de recherche.

Définition 2.19 (***Opérateur de spécialisation***). *Étant donnée une requête* $Q : (\exists Y)(r \wedge L)$ *de variables libres les variables de l'atome de référence R. L'opérateur de spécialisation* $\rho(Q)$ *est l'ensemble des requêtes* $(\exists Y)(r \wedge L \wedge p(t_1, ..., t_n))$ *de variables libres les variables de l'atome de référence telles que :* $p(w_1, ..., w_n)$ *appartient à W et,* $p(t_1, ..., t_n)$ *et* $p(w_1, ..., w_n)$ *sont en accord relativement à Q et W, si pour tout* $i \in \{1...n\}$ *:*

- $w_i =$ " $-$ " $\rightarrow t_i$ *est une variable sans occurrence dans Q,*
- $w_i =$ " $+$ " $\rightarrow t_i$ *est une variable ayant une occurrence dans Q,*
- $w_i =$ " $+ type$" $\rightarrow t_i$ *est une variable de type "type" ayant une occurrence dans Q,*
- $w_i =$ " $- type$" $\rightarrow t_i$ *est une variable de type "type" sans occurrence dans Q,*
- $w_i =$ "a" $\rightarrow t_i$ *est une constante dans le domaine* $dom(i,p)$.

Nous remarquons que c'est une spécialisation par ajout de littéraux en respectant W. L'idée est ainsi d'effectuer des spécialisations successives par l'ajout successif de littéraux en partant de l'atome de référence R.

Exemple 2.6. *Soit un type s et le contexte défini par* $C_3 = \langle p1(x), \{p1(-s), p2(+s,-), p3(+s,-,-)\}\rangle$. *Alors la spécialisation de l'atome de référence p1(x) est définie par les deux requêtes suivantes :* $\rho(p1(x)) \rightarrow \{(\exists y)(p1(x) \wedge p2(x,y)), (\exists w,z)(p1(x) \wedge p3(x,w,z))\}$

Définition 2.20 (***Espace de recherche***). *Étant donné un contexte d'extraction* $C = \langle r, W\rangle$, *l'espace de recherche* $\mathcal{Q}(C)$ *est défini par les spécialisations successives de l'atome de référence :*
$Q(C) = \rho^*(r)$.

La proposition suivante est démontrée dans [31].

Proposition 2.2. *ρ est un opérateur complet relativement à $\mathcal{Q}(C)$.*

Étant donné une requête $Q \in \mathcal{Q}(C)$, on peut noter que pour toute requête $Q' \in \rho(Q)$, on a $Q' \subseteq Q$. D'où le corollaire de la Proposition 2.2 suivant :

Corollaire 2.1. *Soit $C = \langle r, W \rangle$ un contexte. Pour toute requête Q de $\rho(C)$, si Q' est une requête de $\rho(Q)$ alors $Sup(Q'/r, S) \leq Sup(Q/r, S)$.*

Algorithm 3 Algorithme d'évaluation des supports

Entrées: Ensemble de données S ; Ensemble de requêtes Q ; atome de référence r.
Sortie: les supports des requêtes $Q_j \in \mathcal{Q}$.

1: **Pour Tout** $Q_j \in \mathcal{Q}$ **do**
2: $numerateur_{Sup(Q_j)} = 0$
3: **Fin Pour Tout**
4: **Pour Tout** $k_i \in r(S)$ **do**
5: Isoler la portion S_i de S relative à k_i
6: **Fin Pour Tout**
7: **Pour Tout** $Q_j \in \mathcal{Q}$ **do**
8: **Si** $Q_j \theta k_i(S_i)$ est non vide **Alors**
9: incrémenter $numerateur_{Sup(Q_j)}$
10: **Fin Si**
11: **Fin Pour Tout**
12: **Pour Tout** $Q_j \in \mathcal{Q}$ **do**
13: **Retourner** $numerateur_{Sup(Q_j)}/|r(S)j|$
14: **Fin Pour Tout**

L'extraction des requêtes fréquentes peut s'exprimer dans le cadre du formalisme général de Mannila comme suit :$\mathcal{TH}(Q(C), r, S, q) = \{Q \in \mathcal{Q}(C) | q(Q, r, S)$ est vrai dans $S\}$, où $q(Q, r, S)$ est vrai si et seulement si $Sup(Q/r, S) \geq minsup$.

Le calcul de cette théorie se fait par l'algorithme `WARMR` qui est une instance de l'algorithme générique de Mannila et Toivonen [89]. La phase de génération est effectuée par l'intermédiaire de l'opérateur de spécialisation, qui à chaque niveau, permet de générer les requêtes candidates du niveau supérieur.
L'évaluation des supports des candidats se fait par niveau et par valeur de K (l'ensemble des variables de référence).
A un niveau donné, l'algorithme isole les portions S_i de l'ensemble des données relatives aux valeurs k_i des variables de référence, et pour chaque S_i, il incrémente les supports des requêtes candidates Q pour lesquelles $Q\theta k_i(S_i)$ est non vide, où θk_i représente une substitution sur les valeurs de k_i. Cette méthode suppose que les ensembles de données peuvent être stockés en mémoire.
L'algorithme d'évaluation est donné par l'Algorithme 3.
Les ensembles S_i sont caractérisables dans le cas particulier où tout atome d'un motif candidat contient la ou les variables de référence.

Soit $K = (x_1, x_2...x_n)$ le n-uplet des variables de référence $x_1, x_2...x_n$. Pour tout $k_i = (k_{i_1}, k_{i_2}, ..., k_{i_n}) \in dom(1, r), dom(2, r), ..., dom(n, r)$, la portion S_i de S relative à k_i

$$S_i = \bigcup_{L \in W} \{L\theta \in S \mid \theta(x_1) = k_{i_1} \wedge ...\theta(x_n) = k_{i_n}\},$$

où r représente l'atome de référence.
L'exemple suivant illustre le calcul du support grâce à cet algorithme :

Exemple 2.7. *Considérons l'ensemble de faits défini comme suit : S = {p1(1),p1(2),p1(3), p2(1,algo),p2(1,maths),p2(2,algo),p2(2,prog),p2(3,prog),p2(3,se), p3(1,iut,Blois), p3(2,iut,Paris),p3(3,deug,Paris)}, un type t et un contexte* $C_4 = \langle p1(x), \{p1(-t), p2(+t, -), p3(+t, a, a)\}\rangle$.
Considérons maintenant quelques candidats de niveau 2 définis par $\rho(p1(x))$ *:*
$Q_1 : (\exists y)(p1(x) \wedge p2(x, y))$, $Q_2 : p1(x) \wedge p3(x, iut, Blois)$, $Q3 : p1(x) \wedge p3(x, deug, Blois)$, $Q4 : p1(x) \wedge p3(x, deug, Paris)$.

- $S[x = 1] = \{p1(1), p2(1, algo), p2(1, maths), p3(1, iut, Blois)\}$ *: seules* Q_1 *et* Q_2 *seront incrémentées.*
- $S[x = 2] = \{p1(2), p2(2, algo), p2(2, prog), p3(2, iut, Paris)\}$ *: seule* Q_1 *sera incrémentée.*
- $S[x = 3] = \{p1(3), p2(3, prog), p2(3, se), p3(3, deug, Paris)\}$ *: seules* Q_1 *et* Q_4 *seront incrémentées.*

Ainsi, $sup(Q_1/r, S) = 3/3, sup(Q_2/r, S) = 1/3, sup(Q_3/r, S) = 0, sup(Q_4/r, S) = 1/3$

Nous venons de présenter dans la section précédente l'approche `WARMR` qui utilise un biais syntaxique pour restreindre l'espace de recherche. Dans la section suivante, nous allons présenter une autre approche, basée également sur la programmation logique inductive, qui utilise quant à elle une relation de spécialisation entre les requêtes pour une exploration plus efficace de l'espace de recherche.

2.6.2 Approche Programmation Logique Inductive : `WARMER`

Cette approche, baptisée `WARMER` [54] tente d'améliorer l'approche `WARMR` présentée précédemment.
`WARMER` définit une relation de spécialisation entre les requêtes appelée inclusion diagonale (voir Définition 2.21) .
Cette approche parcourt l'espace de recherche grâce à la relation d'inclusion diagonale. En effet, le support est anti-monontone par rapport à cette relation de spécialisation, ce qui permet de parcourir, efficacement l'espace de recherche. La relation d'inclusion diagonale est également utilisée pour éviter, autant que possible, la génération de deux requêtes comparables à une itération donnée.

Définition 2.21 (**Inclusion diagonale**). *Une requête conjonctive* Q_1 *est* ***diagonalement incluse*** *dans une requête* Q_2 *si* Q_1 *est incluse dans une projection de* Q_2.
On note $Q_1 \subseteq^{\Delta} Q_2$

L'exemple suivant illustre la Définition 2.21 :

Exemple 2.8. *Soient les deux requêtes suivantes :*
$Q_1(x) : -aimer(x, y), frequenter(x, z), servir(z, y).$
$Q_2(x, z) : -aimer(x, y), frequenter(x, z), servir(z, y).$
La réponse à la requête Q_1 est l'ensemble des consommateurs qui fréquentent au moins un bar qui sert une bière qu'il aime.
La réponse à la requête Q_2 est l'ensemble des consommateurs et des bars visités servant au moins une bière qu'ils aiment.
Un consommateur peut visiter plusieurs bars servant les bières qu'il préfère, ces bars apparaissent dans la requête Q_2 en même temps que les buveurs. Ainsi, nous avons bien la relation $Q_1 \subseteq^{\Delta} Q_2$.
Ainsi, si Q_2 est non fréquente alors Q_1 est non fréquente.

Remarque : L'inclusion diagonale est basée sur l'inclusion classique entre requêtes.
En effet, si $Q_1 \subseteq Q_2$ alors $Q_1 \subseteq^{\Delta} Q_2$

`WARMER` génère l'ensemble des requêtes conjonctives fréquentes contenant une clé arbitraire R fixée, aussi appelée sujet de comptage, qui permet de définir la forme des motifs qui nous intéressent. Par exemple, si on considère la base de données représentée par la Figure 2.7, on pourrait s'intéresser soit aux buveurs, soit aux bars ou plutôt aux bières séparément.

Cependant, l'approche `WARMER`, à l'instar de la plupart des algorithmes par niveau, doit répondre à deux questions essentielles :
- Comment générer sans redondances de manière efficace les requêtes candidates ?
- Comment évaluer efficacement le support des requêtes candidates générées ?

Génération des candidats :

Dans le but de générer efficacement l'ensemble des requêtes fréquentes, les auteurs ne considèrent que :

1. les règles dont la conséquence contient au moins une variable.
   En effet, le support d'une requête dont la conséquence ne contient pas de variables est au plus égal à un.
2. une seule permutation des variables de la prémisse, en effet, le support de deux requêtes ayant les mêmes variables au niveau de la conséquence et une même prémisse sont équivalentes.

La génération des requêtes fréquentes est basée sur un algorithme par niveau inspiré de l'algorithme générique proposé par Mannila et Toivonen.
Pour chaque requête Q, il est facile de construire une requête Q' telle que $Q \subseteq^{\Delta} Q'$, ceci se fait en ajoutant un atome avec des variables différentes dans le corps de Q et en ajoutant ces mêmes variables au niveau de la tête.
Un autre problème fondamental est la taille très grande de l'espace de recherche, c'est pourquoi, `WARMER` fixe le nombre maximal d'atomes constituant le corps de la requête.
Dans cet espace, les requêtes les plus générales sont clairement définies.
En effet, les requêtes les plus générales sont les requêtes dont :

- la clé est incluse dans la requête ;
- les prémisses ont pour longueur la taille fixée ;
- toutes les variables sont libres ;
- la conséquence contient toutes les variables apparaissant dans la prémisse.

Lors de la phase de génération des requêtes candidates, le successeur d'une requête est considéré que si l'ensemble de ses généralisations est fréquentes. Malheureusement, l'ensemble des généralisations possibles est restreinte par la la limitation du nombre de prédicats dans la prémisse, ce qui rend se test non complet. Les auteurs proposent alors la stratégie suivante : l'ensemble des requêtes candidates est généré, et sont enlevées les requêtes telles qu'il existe une généralisation connue non fréquente ou les requêtes telles que toutes les généralisations ne sont pas dans l'ensemble des requêtes fréquentes du niveau précédent.
Ainsi, après élagage une requête candidate doit être au moins moins générale qu'une requête de l'ensemble des fréquents du niveau précédent.
Les successeurs de chaque requête de l'ensemble des fréquents du niveau précédent sont générés en utilisant l'une des quatre opérations suivantes :

Extension : Ajouter une formule atomique avec de nouvelles variables au corps de la requête.

Jointure : Remplacer les occurrences des variables libre par une autre variable déja présente dans la requête.

Sélection : Remplacer l'occurrence d'une variable libre par une constante.

Projection : Enlever une variable libre apparaissant dans la tête de la requête à condition que la tête ne soit pas vide.

Exemple 2.9. *L'exemple suivant illustre l'application séparée de chacune des opérations définies précédemment sur la requête :* $Q(x, y) : -aimer(x, y), visiter(x, z), servir(z, u)$

Extension : $Q_e(x, y) : -aimer(x, y), visiter(x, z), servir(z, u), aimer(v, w)$

Jointure : $Q_j(x, y) : -aimer(x, y), visiter(x, z), servir(z, y)$, *ici la variable* u *est remplacé par la variable* y.

Sélection : $Q_s(x, y) : -aimer(x, y), visiter(x, z), servir(z, heinenken)$

Projection : $Q_p(x) : -aimer(x, y), visiter(x, z), servir(z, u)$

La proposition suivante 2.3 montre qu'en appliquant une séquence de ces quatre opérations à l'ensemble des fréquents du niveau i, nous obtenons l'ensemble des candidats du niveau $i + 1$.

Proposition 2.3. $Q_1 \subseteq^{\Delta} Q_2$, *si et seulement si,* Q_1 *peut être obtenue à partir de* Q_2 *en appliquant un nombre fini de fois les opérations d'extension, de jointure, de sélection et de projection.*

Preferer	*Buveur*	*Biere*
	alain	carlsberg
	alain	kronenbourg
	augustine	carlsberg
	nicolas	carlsberg
	nicolas	kronenbourg
	nicolas	heinenken

Frequenter	*Buveur*	*Bar*
	alain	objectif lune
	alain	pub saint-hilaire
	augustine	objectif lune
	augustine	pub saint-hilaire
	augustine	la caravane
	nicolas	objectif lune

Servir	*Bar*	*Biere*
	objectif lune	carlsberg
	objectif lune	kronenbourg
	objectif lune	heinenken
	pub saint-hilaire	carlsberg
	pub saint-hilaire	heinenken
	la caravane	kronenbourg

FIGURE 2.7 – Base de données des buveurs

Remarque :

Il faut cependant noter que l'application de cette stratégie génère beaucoup de requêtes redondantes ou équivalentes. De plus, une méthode efficace pour éviter les redondances est inconnue.
Cependant, afin de supprimer les redondances, les auteurs proposent la stratégie suivante : lorsqu'une requête est générée, les auteurs testent si elle n'est pas équivalente à une requête déjà générée, si oui, les opérations précédentes sont appliquées récursivement sur la requête considérée jusqu'à ce qu'elle ne soit plus égale ni équivalente à une requête déjà générée. Cependant, le test d'équivalence de deux requêtes étant un problème *NP-complet*, l'élimination des requêtes redondantes diminue considérablement les performances de l'algorithme proposé.
Après avoir généré les candidats, il faut maintenant tester pour chaque requête candidate si toutes les requêtes plus générales sont fréquentes.
Ceci peut être fait en appliquant l'inverse des quatre opérations. Après la phase de génération, il faut maintenant évaluer les requêtes candidates pour la phase suivante.

Evaluation des requêtes candidates :

L'évaluation des requêtes candidates peut se faire en transformant chaque requête candidate en une *requête SQL*. Cependant, cette méthode est peu réaliste, à cause du nombre important de requêtes à évaluer. Des techniques d'optimisation multi-requêtes existent [106].
Malheureusement, ces techniques ne sont pas implémentées dans les SGBD actuels.
Cependant, il est à remarquer que le support de beaucoup de requêtes peuvent être obtenu à partir du support d'autres requêtes. De ce fait, pour évaluer efficacement des requêtes candidates, seules les requêtes qui satisfont les restrictions suivantes sont considérées :

1. seules les requêtes qui n'ont aucune constante au niveau de la conséquence de la requête sont considérées, car le support des requêtes ayant une constante au niveau de la conséquence est égal au support des requêtes n'ayant pas une constante au niveau de la conséquence mais ayant la même prémisse comme l'illustre l'Exemple 2.10.
2. seules les requêtes qui n'ont pas de variables dupliquées au niveau de la tête sont considérées, car le support de telles requêtes est égal au support des requêtes qui n'ont pas de variables dupliquées au niveau de la prémisse comme l'illustre l'Exemple 2.10.

Exemple 2.10. *Le support de la requête* $Q(alain, z) : -aimer(alain, y), visiter(alain, z), servir(z, heinenken)$ *est égal au support de la requête* $Q_1(z) : -aimer(alain, y), visiter(alain, z), servir(z, heinenken))$.

Le support de la requête $Q_2(alain, z, z) : -aimer(alain, y), visiter(alain, z), servir(z, heinenken)$ *est égal au support de la requête* $Q_3(alain, z) : -aimer(alain, y), visiter(alain, z), servir(z, heinenken))$.

Une autre optimisation proposée par les auteurs est l'évaluation de plusieurs requêtes en même temps. Soit une requête impliquant des constantes, les auteurs ne traitent pas la requête pour chaque variation d'une constante, mais l'ensemble des variations est évalué dans une seule et unique requête globale.
En effet, plusieurs requêtes différenciées par une constante au niveau d'une variable commune, peuvent être traitées toutes ensembles. L'exemple suivant illustre cette optimisation.

Exemple 2.11. *Supposons que la requête*

$$Q(x_1) : -R(x_1, x_2) \tag{2.1}$$

soit fréquente, supposons également que $dom(2, R) = \{1, 2, ..., n\}$. *A partir de la requête* Q, *nous avons au moins les requêtes candidates suivantes en faisant une sélection sur* x_2 *:*

$$\{Q_1(x_1) : -R(x_1, 1), Q_2(x_1) : -R(x_1, 2), Q_3(x_1) : -R(x_1, 3), ..., Q_n(x_1) : -R(x_1, n)\} \tag{2.2}$$

Nous avons alors, un nombre très important de requêtes à évaluer. Cependant, les supports de toutes les requêtes peuvent être obtenus grâce à l'unique requête SQL *suivante :*

```
SELECT x2,COUNT(*)
FROM R GROUP BY x2
HAVING COUNT(*)≥ minsup;
```

Néanmoins, ces optimisations sont loin d'être suffisantes. En effet, un mécanisme de comptage intelligent serait nécessaire afin d'évaluer les supports de requêtes candidates d'un niveau donné en une seule passe sur la base de données afin d'avoir de meilleures performances.

2.6.3 Approche Base de données : `Conqueror`

Dans cette partie, nous allons présenter une approche récente appelée `Conqueror` proposée par Goethals et al [56] et qui permet de trouver l'ensemble des requêtes conjonctives simples (Définition 2.22) fréquentes d'une base de données relationnelle définie sur une base de données quelconque.

Définition 2.22. ***Requêtes conjonctives simples*** *Soit une base de données relationnelle ayant un schéma $R(R_1, R_2, ..., R_n)$ et une instance I de R. Une requête conjonctive simple Q sur R est exprimée de manière algébrique par l'expression suivante : $\pi_X \sigma_F(R_1 \times R_2 \times ... \times R_n)$, avec X un ensemble d'attributs de $R_1 \times R_2 \times ... \times R_n$, F une expression conjonctive de la forme $R_i.A = R_j.B$ ou $R_k.A = c$.*
De plus, X est noté $\pi(Q)$ (projection de la requête Q) et F est noté $\bowtie(Q) \wedge \sigma(Q)$, $\bowtie(Q)$ est la conjonction des conditions de sélection de la forme $R_i.A = R_j.B$ et $\sigma(Q)$ est la conjonction des conditions de sélection de la forme $R_k.A = c$, avec $1 \leq i \leq n, 1 \leq j \leq n$ et $1 \leq k \leq n$, $R_i.A, R_j.B, R_k.A \in U$ et c une constante appartenant au domaine de $R_k.A$.

Remarque : Dans cette définition, R_i apparait exactement une seule fois dans la requête conjonctive simple Q, de plus, chaque relation est considérée comme étant non vide.
`Conqueror` utilise la relation d'inclusion diagonale entre requêtes (voir Définition 2.21) pour parcourir efficacement l'espace de recherche.
Afin de générer des requêtes facilement interprétables, les auteurs ne considèrent pas les requêtes représentant des produits cartésiens. Pour éviter d'avoir à gérer ce type de requêtes, des contraintes supplémentaires sont ajoutées.
Soit une requête Q ayant une jointure $\bowtie(Q)$ constituée d'un ensemble de conditions de sélection de la forme $R_i.A = R_j.A'$, cette conjonction de conditions de sélection induit un partitionnement de U, dans lequel chaque bloc noté β est un ensemble maximal d'attributs tel que pour tout $R_i.A$ et $R_j.A'$ de β, $R_i.A = R_j.A'$ est dans $\bowtie(Q)$. Cette partition est notée $blocs(Q)$.
Egalement, les auteurs proposent la méthode suivante pour déterminer si une partition est associée à un produit cartésien :

- Toute relation est un noeud du graphe.
- Toute égalité entre attributs A et B, appartenant respectivement aux relations R_i et R_j, dans la condition de sélection est considérée comme un arc reliant R_i et R_j.

Ainsi, une requête dont les blocs, définis plus haut, ne sont pas interconnectés représente un produit cartésien.

Exemple 2.12. *Considérons le schéma $\mathcal{D}$, où $sch(R_1) = \{A, B\}$ et $sch(R_2) = \{C, D, E\}$, la requête $Q = \pi_B \sigma_{A=C \wedge C=D}(R_1 \times R_2)$ induit $blocs(Q) = \{\{A, , C, D\}, \{B\}, \{E\}\}$. Ainsi, R_1 et R_2 sont connectées dans $\bowtie(Q)$. De ce fait, Q n'est pas une requête de type produit cartésien. Considérons maintenant la requête $Q_1 = \pi_A D \sigma_{A=C \wedge E=c}(R_1 \times R_2)$, Q_1 est une requête telle que $\bowtie(Q_1) = (A = C)$, $\pi(Q_1) = AD$, $\sigma(Q_1) = (E = e)$.*
$blocs(Q_1)$ contient alors quatre blocs : {{A,C},{B},{D} et {E}} et nous pouvons remarquer que R_1 et R_2 ne sont pas connectées dans $\bowtie(Q_1)$. Ainsi, Q_1 est une requête de type produit cartésien.

L'Algorithme `Conqueror`

L'algorithme `Conqueror` (`Conj`onctive `Query` `Generator`) est divisé principalement en deux phases comme l'algorithme générique de Mannila et Toivonen.
Ces deux phases sont la génération et l'évaluation des requêtes candidates :

Génération des requêtes candidates :

Cette phase s'effectue grâce à la génération de l'ensemble des instanciations possibles des ensembles d'attributs de projection et de jointure des requêtes conjonctives fréquentes du niveau précédent en utilisant une stratégie en largeur d'abord.
Les trois étapes pour générer les requêtes candidates sont : la boucle de jointure, la boucle de projection et la boucle de sélection.
Dans ce qui suit, nous allons détailler chacune de ces trois étapes :

Boucle de Jointure :

Dans cette boucle, un ensemble de partitions des attributs de sélection des requêtes est généré. La génération des partitions d'un ensemble donné est un problème connu et admet des solutions efficaces comme la solution "Restricted Growth String" proposée dans [119], c'est cette solution qui est utilisée par les auteurs.
Une "Restricted Growth String" est un tableau $a[1 \dots n]$, où n est le nombre d'attributs à partitionner et $a[i]$ est l'identifiant d'un bloc de la partition dans lequel l'attribut i est présent.
Une partition est représentée par une chaine de caractère définie de la manière suivante :
$a[1] = 1$ et $\forall i \in \{1, 2, \dots n-1\}, a[i+1] \leq 1 + max(a[1], a[2], \dots, a[i])$.

Exemple 2.13. *Soit $A_1, A_2, A_3, \dots, A_n$ l'ensemble des attributs présents dans la base. La chaine "1221" représente la condition conjonction d'égalités $A_1 = A_4$ et $A_2 = A_3$, qui correspond à la partition $\{\{A_1, A_4\}, \{A_2, A_3\}\}$.*

L'Algorithme 4 décrit la boucle de génération des partitions représentant les jointures. Cet algorithme débute avec la chaine "1" qui représente la partition la plus générale c'est à dire celle qui contient l'ensemble des attributs. Soit une chaine de caractères représentant une partition, les partitions les moins générales sont générées en ajoutant un attribut dans un bloc existant.
Pour être sûr que les partitions ne sont pas dupliquées, un ordre sur les attributs est défini et un attribut est ajouté si son ordre est supérieur à l'ordre de tous les attributs présents dans la partition.
Par exemple pour quatre attributs, l'espace complet des différents blocs générés est représenté par la Figure 2.8.
L'Algorithme 5 décrit l'algorithme principal *Conqueror*.
Comme nous l'avons mentionné précédemment les produits cartésiens ne sont pas considérés ainsi, cet Algorithme teste tout d'abord (cf. ligne 5) si la "Restricted Growth String" associée à la requête courante représente un produit cartésien, si le test est négatif les successeurs sont générés grâce à la boucle projection suivante sinon la boucle de projection n'est pas exécutée.

Algorithm 4 RestrictedGrowthString(String prefix,Length m)

```
1: list ← {}
2: last ← length(prefix)
3: Si last < m Alors
4:    for i = 1 à m do
5:       max ← max({prefix[j]|0 ≤ j < m})
6:       nprefix ← prefix
7:       Si i > last Alors
8:          for k = last à i − 1 do
9:             max ← max + 1
10:            nprefix[k] ← max
11:         Fin Pour Tout
12:      Fin Si
13:      for k = 1 à max do
14:         nprefix[i] ← l
15:         ajouter(list, nprefix)
16:      Fin Pour Tout
17:   Fin Pour Tout
18: Fin Si
19: Retourner list
```

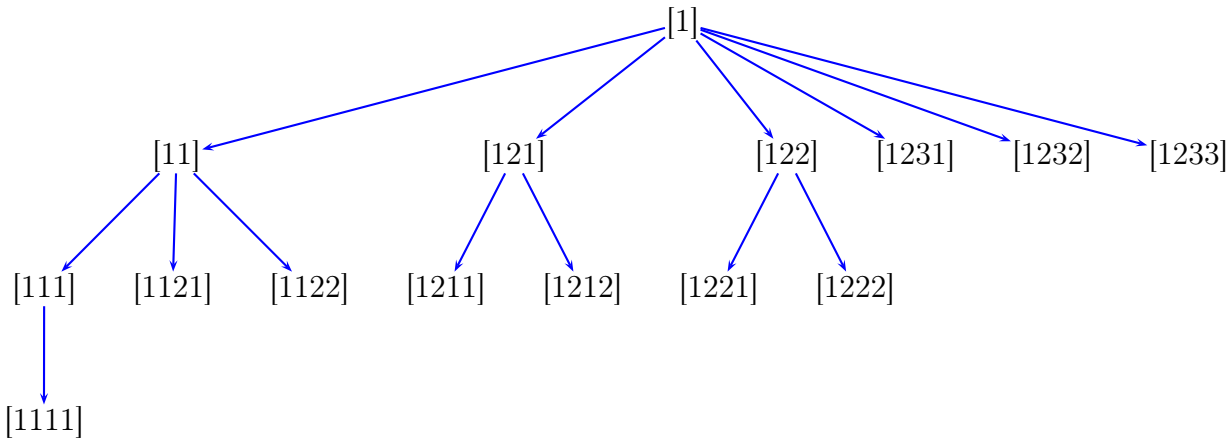


FIGURE 2.8 – Espace de recherche des blocs pour 4 attributs

Algorithm 5 Conqueror

Entrées: F L'ensemble des requêtes fréquentes.
Sortie: L'ensemble $\mathcal{TH}(\varphi, S, q)$.

1: $rgs \leftarrow$ "1" {initial restricted growth}
2: $push(Queue, Q)$
3: **Tant Que** $Queue$ non vide **do**
4: $SQ \leftarrow pop(Queue)$
5: **Si** rgs ne représente pas un produit cartésien **Alors**
6: $F \leftarrow F \cup ProjectionLoop(SQ)$
7: **Fin Si**
8: $children \leftarrow RestrictedGrowth(\sigma(SQ), m)$
9: **Pour Tout** rgs dans children **do**
10: **Si** la sélection définie par rgs n'est pas moins générale que la sélection la moins générale **Alors**
11: $\sigma(SQC) \leftarrow rgs$
12: $push(Queue, SQC)$
13: **Fin Si**
14: **Fin Pour Tout**
15: **Fin Tant Que**
16: **Retourner** F

Boucle de Projection :

Pour chaque requête obtenue grâce à la boucle de jointure précédente, les projections sont générées grâce à l'Algorithme 6.
L'algorithme génère, grâce à la relation d'inclusion diagonale, les successeurs des requêtes considérées.
Ceci se fait en générant les sous-ensembles des attributs de projection des requêtes considérées en utilisant les propriétés de l'inclusion diagonale. Il faut noter, que tous les successeurs possibles ne sont pas générés afin d'éviter la génération de requêtes équivalentes. En effet, toute suppression d'un attribut appartenant au même bloc génère des requêtes équivalentes. Ainsi, si plusieurs attributs appartiennent à un même bloc seul l'un d'entre eux est supprimé.
L'évaluation des supports est également effectuée dans la boucle de projection (cf. ligne 6) de l'Algorithme 6.

Boucle de sélection :

Cette boucle permet de générer les successeurs des requêtes obtenues à l'étape précédente, grâce à l'assignation de constante aux attributs de jointure des requêtes considérées. Ces assignations sont de la forme $R_k.A = c$ (c'est à dire les égalités entre attributs et constantes).

Dans cette boucle de sélection, pour éviter la génération de requêtes équivalentes, les auteurs évitent l'instanciation de deux attributs appartenant au même bloc avec une même constante comme l'illustre l'Exemple 2.14 .

Algorithm 6 ProjectionLoop

Entrées: Une requête conjonctive simple Q.

Sortie: F_Q l'ensemble des requêtes fréquentes obtenue grâce à une projection sur Q et ayant pour jointure celle de Q.

1: $\pi(q) \leftarrow blocs(Q)$ {tous les blocs connectés de $\bowtie$ (Q)}
2: $push(Queue, Q)$
3: **Tant Que** $Queue$ non vide **do**
4: $PQ \leftarrow pop(Queue)$
5: **Si** monotonie(PQ) **Alors**
6: $support(PQ) \leftarrow EvaluateSupport(PQ)$
7: **Si** $support(PQ) > minsup$ **Alors**
8: $F_Q \leftarrow F_Q \cup ConstantLoop(PQ)$
9: $removed \leftarrow blocs(PQ) \notin \pi(PQ)$
10: $torem \leftarrow blocs(PQ) > last$ of removed //les blocs sont supposés ordonnés dans le but d'une génération sans redondance
11: **Pour Tout** p_i dans $torem$ **do**
12: $\pi(PQC) \leftarrow \pi(PQ)$ avec le bloc p_i supprimé
13: $push(Queue, PQC)$
14: **Fin Pour Tout**
15: **Fin Si**
16: **Fin Si**
17: **Fin Tant Que**
18: **Retourner** F_Q

Exemple 2.14. *Soit la base de données D tel que* $sch(R_1) = \{A, B\}, sch(R_2) = \{C, D, E\}$ *et* $sch(R_3) = \{F\}$*, il est facile de voir que les requêtes* $\pi_A\sigma_{B=C\wedge C=a}(R_1 \times R_2 \times R_3)$ *et* $\pi_A\sigma_{B=C\wedge B=a}(R_1 \times R_2 \times R_3)$ *sont équivalentes. De même,* $\pi_A\sigma_{B=C\wedge C=a}(R_1 \times R_2 \times R_3)$ *et* $\pi_A C\sigma_{B=C\wedge C=a}(R_1 \times R_2 \times R_3)$ *sont équivalentes.*

L'Algorithme 7 décrit la boucle de sélection. Cette algorithme génére les requêtes de sélection candidates à partir d'une requêtes Q donnée en utilisant une stratégie de parcours en largeur d'abord.
Ce processus est répété jusqu'à ce qu'il ne soit plus possible de générer une nouvelle requête par spécialisation (ligne 8).
Ici également, pour ne pas générer des requêtes redondantes, cet algorithme n'alloue pas de constante à des blocs qui sont dans le schéma de projection comme l'illustre l'item 2 de l'exemple 2.14,

Algorithm 7 ConstantLoop

Entrées: Une requête conjonctive Q.
Sortie: F_Q l'ensemble des requêtes fréquentes avec la même jointure et la même projection que Q.

1: $push(Queue, Q)$
2: **Tant Que** $Queue$ non vide **do**
3: $CQ \leftarrow pop(Queue)$
4: **Si** $\sigma(PQ) = \emptyset$ **Alors**
5: $toadd \leftarrow$ tous les blocs de $blocs(CQ) \notin \pi(CQ)$
6: **Sinon**
7: $uneq \leftarrow$ tous les blocs de $blocs(CQ) \notin \sigma(CQ) \cup \pi(CQ)$
8: $toadd \leftarrow$ blocs dans $uneq > last$ de $\sigma(CQ)$ //On suppose que les blocs sont ordonnés
9: **Fin Si**
10: **Pour Tout** B_i dans $toadd$ **do**
11: $\sigma(CQC) \leftarrow \sigma(CQC) \cup B_i$
12: **Si** existe une constante fréquente pour $\sigma(CQC)$ dans la base **Alors**
13: $F_Q \leftarrow F_Q \cup CQC$
14: $push(Queue, CQC)$t
15: **Fin Si**
16: **Fin Pour Tout**
17: **Fin Tant Que**
18: **Retourner** F_Q

Evaluation des requêtes candidates : Dans le but d'évaluer chaque requête générée, *Conqueror* évalue leur support par rapport à l'instance de base de données en les transformant en requête SQL. Chaque requête de la forme $\pi_X\sigma_F(R_1 \times R_2 \times ... \times R_n)$ est transformée en la requête SQL suivante :

```
SELECT COUNT(DISTINCT X)
FROM R1, R2, ..., Rn
WHERE F
```

Il est clair que l'algorithme nécessite l'évaluation de plusieurs requêtes ayant la même jointure dans les boucles de sélection et projection, c'est la raison pour la laquelle, quelques optimisations sont proposées par les auteurs.

En effet, seules les requêtes plus générales (les requêtes contenant les projections les plus générales et ne contenant pas de conditions de sélection constante) sont transformées en requêtes SQL pour être évaluées et stockées dans une table temporaire τ, ainsi, les autres extractions seront déduites de ces tables temporaires.

Les requêtes moins générales sont réécrites en utilisant les résultats des requêtes les plus générales comme l'illustre l'exemple suivant :

Soient deux requêtes $Q = \pi_X \sigma_F(R_1 \times R_2 \times ... \times R_n)$ et $Q' = \pi_{X'} \sigma_F(R_1 \times R_2 \times ... \times R_n)$, avec $X' \subseteq X$, si τ est le résultat de la requête Q.

Q' sera évaluée de la manière suivante :

```
SELECT COUNT(DISTINCT X')
FROM τ
```

Pour évaluer les requêtes mettant en jeu l'égalité d'un attribut et d'une constante d'autres techniques d'optimisation sont développées. En effet, il est difficile et inefficace d'évaluer une à une chaque requête mettant en jeu l'égalité d'un attribut et d'une constante de la base de données.

Soient les requêtes de la forme $Q'' = \pi_{X'} \sigma_{\bowtie(Q) \wedge A=?}(R_1 \times R_2 \times ... \times R_n)$, avec $X' \subseteq X$, il est évident que les requêtes définies par Q'' sont moins générales que celle définies par les requêtes de type Q, les requêtes de cette forme seront évaluées en utilisant la table τ grâce à l'unique requête suivante :

```
SELECT A AS COUNT(*) AS sup
FROM τ
GROUP BY A
HAVING COUNT(*)>= minsup
```

Les résulats de cette requête sont stockées dans une nouvelle table (τ_A). Cette table temporaire contient alors l'ensemble des requêtes de la forme $\pi_{X'} \sigma_{\bowtie(Q) \wedge A=v}(R_1 \times R_2 \times ... \times R_n)$ (avec v appartenant au domaine de A) fréquentes ainsi que leur support.

Dans la suite, ces tables temporaires sont combinées pour évaluer des requêtes mettant en jeu plusieurs conditions de sélection comme l'illustre l'exemple suivant :

Exemple 2.15. *Soit τ_A et τ_B deux tables temporaires contenant le support des requêtes fréquentes respectivement de la forme $\pi_{X'} \sigma_{\bowtie(Q) \wedge A=v}(R_1 \times R_2 \times ... \times R_n)$, $\pi_{X'} \sigma_{\bowtie(Q) \wedge B=v_1}(R_1 \times R_2 \times ... \times R_n)$ ainsi que leur support (avec v et v_1 des constantes appartenant au domaine de A et B). Les auteurs proposent l'optimisation suivantes pour évaluer les requêtes la forme $\pi_{X'} \sigma_{\bowtie(Q) \wedge A=? \wedge B=?}(R_1 \times R_2 \times ... \times R_n)$ grâce à τ, τ_A et τ_B de la manière suivante :*

```
SELECT A,B AS COUNT(*)
FROM (SELECT A, B, X'
                                FROM τ NATURAL JOIN
                                (SELECT *FROM
                                (SELECT A FROM τ_A)
                                NATURAL JOIN
                                (SELECT B FROM τ_B)
)
)
GROUP BY A,B
HAVING COUNT(*)>= minsup
```

Le résultat de cette requête est ensuite sauvegardé dans une nouvelle table temporaire $\tau_{A,B}$ et cette table est utilisé de la même manière pour évaluer les requêtes ayant plus de deux deux conditions de sélection.

Ces optimisations permettent d'améliorer l'évaluation des requêtes candidates, mais une méthode pour évaluer les requêtes d'un niveau donné en une seule passe serait plus efficace que l'utilisation de requêtes SQL à cause des E/S avec la base de données. Dans notre contribution, nous proposerons une méthode pour l'évaluation des requêtes en une seule passe.

Elagage des requêtes Candidates :

L'évaluation des candidats avec des requêtes SQL est coûteuse, car les requêtes SQL sont envoyées à la base de données puis le résultat est récupéré. Dans le but de diminuer les E/S avec la base de données, `Conqueror` élague les requêtes générées qui ont une requête plus générale non fréquente, en utilisant l'anti-monotonie du support par rapport à la relation de contenance diagonale (cf. ligne 5).
L'Algorithme 8 décrit la fonction de test de la monotonie. Cette algorithme permet de tester si l'ensemble des généralisations d'une requête candidate données sont fréquentes, si le test est positif cette algorithme retourne vrai sinon elle retoure faux.
Il permet ainsi d'élaguer les requêtes candidates non fréquentes grâce à la propriété d'anti-monotonie du support par rapport à l'inclusion diagonale. Cette fonction est utilisée à la ligne 6 de l'Algorithme 6.
Pour une requête Q, l'algorithme regarde si toutes requêtes Q', telles que $Q \subset_\Delta Q'$ et qu'il n'existe pas de requête Q'' vérifiant $Q \subset_\Delta Q'' \subset_\Delta Q'$, sont fréquentes.
Le calcul des Q' se fait en utilisant l'inverse des opérations de Jointure, de Projection et de Sélection (ligne 6, 8-12, 15-20).
Naturellement, l'application de l'opération inverse de la jointure peut générer des requêtes de type produit cartésien. Comme de telles requêtes ne sont pas considérés dans `Conqueror`, l'algorithme ignore tout simplement les requêtes les plus générales de type produit cartésien (ligne 21).

Algorithm 8 Monotonie

Entrées: Une requête Conjonctive Q.
Sortie: Vrai ou Faux.

Pour Tout bloc $\beta \in blocs(Q) \notin \pi(Q)$ **do**
  $PP \leftarrow PP \cup \{Q'\}$ avec $\pi(Q') = \pi(Q) \cup \beta, \bowtie (Q') = \bowtie (Q)$ and $\sigma(Q') = \sigma(Q)$
Fin Pour Tout
Pour Tout bloc $\beta \in blocs(Q)$ **do**
  $SP \leftarrow SP \cup \{Q'\}$ avec $\sigma(Q') = \pi(Q) \setminus \beta, \bowtie (Q') = \bowtie (Q)$ and $\pi(Q') = \pi(Q)$
Fin Pour Tout
Pour Tout bloc $\beta \in blocs(Q)$ **do**
  Pour Tout splits de $\beta \in \beta_1$ et β_2 **do**
    $\bowtie (Q') = (\bowtie (Q) \setminus \beta) \cup \{\beta_1, \beta_2\}$
    Si $\beta \in \pi(Q)$ **Alors**
      $\sigma(Q') = \sigma(Q)$
      $JP \leftarrow JP \cup \{Q'\}$ avec $\pi(Q') = (\pi(Q) \setminus \beta) \cup \beta_1$
      $JP \leftarrow JP \cup \{Q'\}$ avec $\pi(Q') = (\pi(Q) \setminus \beta) \cup \beta_2$
      $JP \leftarrow JP \cup \{Q'\}$ avec $\pi(Q') = (\pi(Q)$
    Sinon
      $\pi(Q') = \pi(Q)$
      Si $\beta \in \sigma(Q)$ **Alors**
        $JP \leftarrow JP \cup \{Q'\}$ avec $\pi(Q') = (\pi(Q) \setminus \beta) \cup \beta_1$
        $JP \leftarrow JP \cup \{Q'\}$ avec $\pi(Q') = (\pi(Q) \setminus \beta) \cup \beta_2$
      Sinon
        $JP \leftarrow JP \cup \{Q'\}$ avec $\sigma(Q') = \sigma(Q)$
      Fin Si
    Fin Si
  Fin Pour Tout
Fin Pour Tout
Pour Tout $MCG \in (JP \cup PP \cup SP)$ **do**
  Si MCG n'est pas un produit cartésien **Alors**
    Si $support(MCG) < minsup$ **Alors**
      Retourner $false$
    Fin Si
  Fin Si
Fin Pour Tout
Retourner $true$

Résultats expérimentaux :

Les auteurs ont effectué différents tests en utilisant des extraits des bases de données ImDB [2] et QuizDB [3], dont les caractéristiques sont résumées par la figure 2.9.
Les figures 2.10 et 2.11 montrent l'évolution des temps d'exécution en fonction du support, comme nous pouvons le remarquer, le temps d'exécution augmente rapidement quand le support diminue car le nombre de requêtes candidates pour chaque niveau augmente.
De plus, nous remarquons que `Conqueror` permet de découvrir l'ensemble des requêtes fréquentes des bases ImDB et QuizDB en des temps raisonnables (au maximum 776 s).
Cependant, comme nous le verrons dans la suite, ces temps peuvent être améliorés lorsque l'on prend en compte les dépendances fonctionnelles et les dépendances d'inclusion présentes dans les bases de données.

attribute	#values
actors.*	45342
actors.aid	45342
actors.name	45342
genres.*	21
genres.gid	21
genres.name	21
movies.*	71912
movies.mid	71912
movies.name	71906
actormovies.*	158441
actormovies.aid	45342
actormovies.mid	54587
genremovies.*	127115
genremovies.gid	21
genremovies.mid	71912

(a) IMDB

attribute	#values
scores.*	868755
scores.score	14
scores.player	31934
scores.qid	5144
scores.date	862769
scores.results	248331
scores.month	12
scores.year	6
quizzes.*	4884
quizzes.qid	4884
quizzes.title	4674
quizzes.author	328
quizzes.category	18
quizzes.language	2
quizzes.number	539
quizzes.average	4796

(b) QuizDB

FIGURE 2.9 – Caractéristiques des bases IMDB et QuizDB

2.6.4 Approche Base de données : `IncMiner`

L'idée de concevoir un système d'extraction de connaissances interactif, multi utilisateurs et multi tables a été proposée par Diop et al. ([36]). Cette idée est fondée sur l'algèbre relationnelle. Il s'agit d'offrir aux utilisateurs le moyen de modifier et d'affiner les tâches (requêtes) d'extraction de règles sur plusieurs tables. Le processus d'extraction de règles possède un aspect incrémental vu que non seulement les réponses aux tâches exécutées sont sauvegardées mais il est également possible de définir de nouvelles tâches par composition des tâches déjà accomplies.
En effet, un utilisateur, après une première tâche d'extraction, peut décider de raffiner celle-ci, ou de formuler une tâche d'extraction qui peut être déduite des tâches d'extraction précédentes. Il en est de même lorsque plusieurs utilisateurs effectuent une extraction à partir des mêmes données. L'idée de cette approche est illustrée par l'Exemple 2.16.

2. http ://www.imdb.com
3. http ://www.persecondewijzer.net

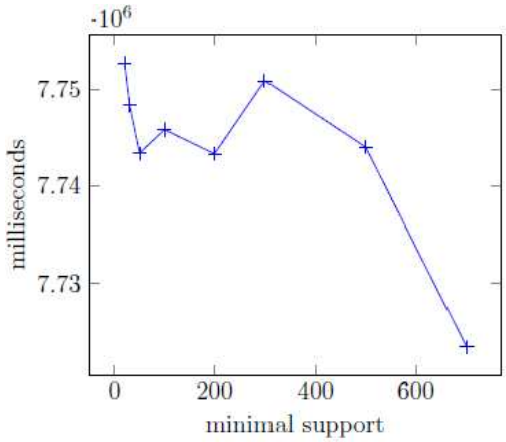


FIGURE 2.10 – Evolution du temps en fonction du support pour ImDB

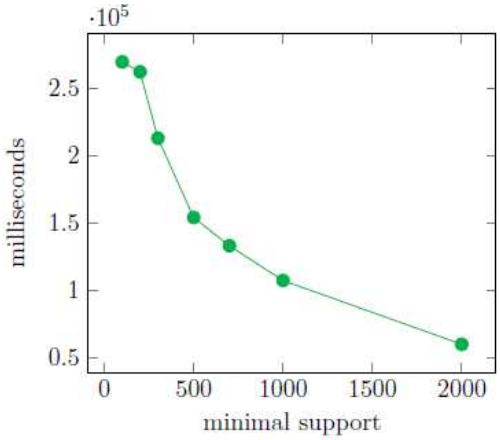


FIGURE 2.11 – Evolution du temps en fonction du support pour QuizDB

Cette approche est basée sur la notion de contexte d'extraction. Un contexte d'extraction $\mathcal{C}$ est un triplet $\mathcal{C} = \langle R, \mathcal{B}, \Sigma \rangle$:

- R est une requête de l'algèbre relationnelle qui représente la référence du contexte d'extraction et a pour résultat l'ensemble des objets qui intéressent l'utilisateur,
- $\mathcal{B}$ est la base du contexte d'extraction : c'est une requête donnant l'ensemble des attributs caractérisant les objets qui intéressent l'utilisateur,
- Σ est le domaine d'intérêt du contexte d'extraction : elle est constituée d'un ensemble de condition de sélection élémentaire sur la base de contexte. Ces conditions sont de la forme $(A = v)$, ou A est un attribut de $\mathcal{B}$ parmi les attributs de r, et v est une valeur du domaine de A.

Etant donné un contexte $\mathcal{C} = \langle R, \mathcal{B}, \Sigma \rangle$, l'espace de recherche qui lui est associé est défini par :

1. Σ^*, ensemble de toutes les conditions de sélection de la forme $(A_1 = v_1) \wedge \cdots \wedge (A_n = v_n)$, tel que pour tout $i \in [1..n], (A_i = v_i) \in \Sigma$.

2. Q($\mathcal{C}$) représente l'ensemble de toutes les requêtes candidates de C, de la forme $q = R \bowtie \sigma_S(\mathcal{B})$, avec $S \in \Sigma^*$.
3. R($\mathcal{C}$) représente l'ensemble de toutes les règles de la forme $q_1 \rightarrow q_2$, ou $q_1 = R \bowtie \sigma_{S_1}(\mathcal{B})$ et $q_2 = R \bowtie \sigma_{S_1 \wedge S_2}(\mathcal{B})$. Une règle est satisfaite par une instance I si $\pi_K(q_1))(I) \subseteq (\pi_K(q_2))(I)$, avec K représentant l'ensemble des attributs de la requête référence.

Le critère de qualité des requêtes est comme dans le cas classique le nombre de tuples retournés. Comme dans un scénario classique d'extraction de règles d'association, l'extraction étant donnée une tâche d'extraction, s'effectue en deux phases : recherche de toutes les requêtes fréquentes, puis génération de toutes les règles intéressantes à partir des requêtes fréquentes. La recherche des requêtes fréquentes étant la phase la plus coûteuse en temps et espace mémoire, les auteurs de ce travail proposent d'accélérer cette recherche en utilisant les résultats sauvegardés des tâches d'extraction antérieures par composition de contextes d'extraction. Pour cela, ils ont défini et étudié un ensemble d'opérations sur les contextes similaires aux opérations de l'algèbre relationnelle, il ont également démontrés des propriétés intéressantes sur ces opérateur. Nous allons présenter ces opérateurs, $freq_\alpha(\mathcal{C}', I))$ représentera les requêtes fréquentes du contexte $\mathcal{C}'$ par rapport à l'instance I.

Projection :

Si $\mathcal{C} = \langle R, \mathcal{B}, \Sigma \rangle$ est un contexte, et X un ensemble d'attributs. Si $att(\Sigma) \subseteq X \bowtie sch(\mathcal{B})$, alors la b-projection de $\mathcal{C}$ sur X, notée $\pi^b_X(\mathcal{C})$ est définie par $\pi^b_X(\mathcal{C}) = \langle R, \pi_X(\mathcal{B}), \Sigma \rangle$.

Sélection :

Si $\mathcal{C} = \langle R, \mathcal{B}, \Sigma \rangle$ un contexte, et S une condition de sélection de Σ^*. Si $att(S) \subseteq sch(\mathcal{B})$, la b-selection de $\mathcal{C}$ sur S, notée $\sigma^b_S(\mathcal{C})$ est définie par $\sigma^b_S(\mathcal{C}) = \langle R, \sigma_S(\mathcal{B}), \Sigma \rangle$.

Jointure :

Si $\mathcal{C}_1 = \langle R, \mathcal{B}_1, \Sigma_1 \rangle$ et $\mathcal{C}_2 = \langle R, \mathcal{B}_2, \Sigma_2 \rangle$ sont deux contextes de même référence R. La jointure de $\mathcal{C}_1$ et $\mathcal{C}_2$, notée $\mathcal{C}_1 \bowtie \mathcal{C}_2$, est définie par : $\mathcal{C}_1 \bowtie_b \mathcal{C}_2 = \langle R, \mathcal{B}_1 \bowtie \mathcal{B}_2, \Sigma_1 \cup \Sigma_2$.

Union, Intersection et Différence :

Si $\mathcal{C}_1 = \langle R, \mathcal{B}_1, \Sigma_1 \rangle$ et $C_2 = \langle R, \mathcal{B}_2, \Sigma_2 \rangle$ sont deux contextes de même référence R et de même domaine Σ, tels que $sch(B_1) = sch(B_2)$.

1. L'union de $\mathcal{C}_1$ et $\mathcal{C}_2$, notée $\mathcal{C}_1 \cup \mathcal{C}_2$ est définie par $\mathcal{C}_1 \cup_b \mathcal{C}_2 = \langle R, \mathcal{B}_1 \cup \mathcal{B}_2, \Sigma_1 \rangle$.
2. L'intersection de $\mathcal{C}_1$ et $\mathcal{C}_2$, notée $\mathcal{C}_1 \cap_b \mathcal{C}_2$ est définie par $\mathcal{C}_1 \cap_b \mathcal{C}_2 = \langle R, \mathcal{B}_1 \cap \mathcal{B}_2, \Sigma_1 \rangle$.
3. La différence de $\mathcal{C}_2$ et $\mathcal{C}_1$, notée $\mathcal{C}_2 \setminus_b \mathcal{C}_1$ est définie par $\mathcal{C}_2 \setminus_b \mathcal{C}_1 = \langle R, \mathcal{B}_2 - \mathcal{B}_2, \Sigma_1 \rangle$..

Restriction :

Si $\mathcal{C} = \langle R, \mathcal{B}, \Sigma \rangle$ un contexte, avec $K = sch(R)$, et soit S une condition de sélection dans Σ^*. La restriction de C sur S est définie par : $\rho_S(\mathcal{C}) = \langle \pi_K(R \bowtie \sigma_S(\mathcal{B})), \sigma_S(\mathcal{B}), \Sigma \rangle$. à une requête de $freq_\alpha(\rho_S(C), I)$.

De plus, les auteurs montrent les proprétés suivantes, pour tout seuil de support α et toute instance I :

- $freq_\alpha(\pi^b_X(\mathcal{C}), I) \sim \pi_X(freq_\alpha(\mathcal{C}, I))$
- toute requête de $freq_\alpha(\sigma^b_S(\mathcal{C}), I)$ est équivalente à une requête de $\sigma_S(freq_\alpha(\mathcal{C}, I))$
- $freq_\alpha(\mathcal{C}_1 \bowtie_b \mathcal{C}_2, I)$ est équivalente à une requête de $freq_\alpha(\mathcal{C}_1, I)) \bowtie freq_\alpha(\mathcal{C}_2, I))$
- $freq_\alpha(\mathcal{C}_i, I) \subseteq freq_\alpha(\mathcal{C}_1 \cup \mathcal{C}_2, I)$, pour i=1,2.
- $freq_\alpha(\mathcal{C}_1 \cap \mathcal{C}_2, I) \subseteq freq_\alpha(C_i, I)$, pour i=1,2.
- $freq_\alpha(\mathcal{C}_2 \setminus_b \mathcal{C}_1, I) \subseteq freq_\alpha(C_2, I)$.
- $\rho_S(freq_\alpha(\mathcal{C}), I)$ est équivalente à une requête de $freq_\alpha(\rho_S(C), I)$.

Ces différentes propriétés permettent ainsi de déduire les tâches d'extraction futures à partir de tâches déja effectuées. L'exemple 2.16 donne une idée de cette approche.

Exemple 2.16. *Soit une base de données de ventes, définie par les 4 relations suivantes :*

- *$Client(Cid, Cprof, Caddr)$, où Cid, $Cprof$ et $Caddr$ sont respectivement les identifiants, professions, et adresses des clients,*
- *$Produit(Pid, Pype)$, où Pid et $Ptype$ sont respectivement les identifiants, et types de produits,*
- *$Magasin(Mid, Mnom, Maddr)$ où Mid, $Mnom$, et $Maddr$ sont respectivement les identifiants, les noms, et adresses des magasins,*
- *$Vente(Cid, Pid, Mid, date)$ donne les relations entre Client, Produit et Magasin,*

un tuple $\langle c, p, m, d \rangle$ de cette table exprime que le client d'identifiant c a achète un produit identifié par p dans le magasin identifié par m à la date d.
Soit C_1 un exemple de contexte d'extraction sur Vente. $\mathcal{C}_1 = \langle R_1, \mathcal{B}_1, \Sigma_1 \rangle$, où :

- *$R_1 = tousClients$,*
- *$\mathcal{B}_1 = Client \bowtie Vente \bowtie Produit$,*
- *$\Sigma_1 = \{Cprof = chercheur, Cprof = avocat, Ptype = lait, Ptype = biere\}$.*

Etant donné C_1, la règle suivante est un exemple de règle d'association pouvant être considérée : $tousClients \bowtie \sigma_{Cprof=chercheur}(\mathcal{B}_1) \rightarrow tousClients \bowtie \sigma_{Ptype=biere}(\mathcal{B}_1)$.
Cette règle exprime le fait que si un client est un chercheur, alors il achète de la bière. En d'autres termes, cette règle est satisfaite dans une instance I de Ventes, si :

$\pi_{Cid}(tousClients \bowtie \sigma_{Cprof=chercheur}(\mathcal{B}_1))(I) \subseteq (\pi_{Cid}(tousClients \bowtie \sigma_{Ptype=biere}(\mathcal{B}_1))(I)$.

Etant donné un nouveau contexte $\mathcal{C}^*_1$, soit $q^*_1 = tousClients \bowtie \sigma_S(\sigma_{Caddr=Paris}(\mathcal{B}_1))$, une requête de C^*_1. Nous remarquons que la réponse à la requête q^*_1 est incluse dans la réponse de $q_1 = tousClients \bowtie \sigma(\mathcal{B}_1)$.
De façon plus générale, toute requête q^*_1 de $\mathcal{C}^*_1$ est contenue dans une requête q_1 de $\mathcal{C}_1$. Il en découle que le support de n'importe quelle requête q^*_1 de C^*_1 est inférieur au support d'une requête q_1 de $\mathcal{C}_1$.
Par conséquent, si une requête q^*_1 de $\mathcal{C}^*_1$ n'est pas incluse dans une requête q_1 de $\mathcal{C}_1$ alors elle ne peut être fréquente.
Ainsi, si toutes les requêtes de $\mathcal{C}_1$ sont gardées, il sera possible d'éliminer quelques requêtes

de $\mathcal{C}_1^*$, avec l'intérêt sous-jacent de ne pas accéder à la base de données, ce qui allège considérablement la recherche des requêtes fréquentes.
Supposons que l'on s'intéresse aux règles reliant les professions des clients aux noms de magasins où ils ont effectué leurs achats.
Si on s'intéresse encore aux professions de chercheur et avocat, et qu'en plus on se focalise sur les magasins Carrefour et Ikea, alors on définit un contexte $\mathcal{C}_2 = \langle R1, \mathcal{B}2, \Sigma_2 \rangle$ tel que $R_1 = tousClients$, $\mathcal{B}_2 = Client \bowtie Vente \bowtie Magasin$ et $\Sigma_2 = \{Cprof = chercheur, Cprof = avocat, M_{nom} = Carrefour, M_{nom} = Ikea\}$.
Supposons que les requêtes fréquentes de $\mathcal{C}_2$ sont calculées et sauvegardées. Alors, en utilisant $\mathcal{C}_1$ et $\mathcal{C}_2$, il est possible de calculer le contexte $\mathcal{C}_{12}$ défini par le triplet $\langle tousClients, \mathcal{B}_1 \bowtie \mathcal{B}_1, \Sigma_1 \bowtie \Sigma_2 \rangle$.
Ce nouveau contexte permet de considérer les relations entre les professions des clients, les types de produits qui ils achètent, et les noms des magasins où ils effectuent leurs courses. Le calcul des requêtes fréquentes de $\mathcal{C}_{12}$ peut être optimisé grâce aux requêtes fréquentes issues de $\mathcal{C}_1$ et celles de $\mathcal{C}_2$.

Résultats expérimentaux :

L'efficacité de cette approche incrèmentale a été démontrée par ses auteurs grâce au développement du système `IncMiner` ainsi qu'aux différentes expérimentations menées.
En effet, les auteurs ont effectué plusieurs tests sur des bases de données synthétiques, en utilisant, un générateur basé sur celui utilisé dans [8]. Il faut noter cependant, que ce dernier a été modifié pour gérer le cas multi-tables.
Les tests sont effectués sur une base de données générique, appelée $DBTest$ qui contient 3 tables ayant respectivement pour schémas $Cust(x_1, x_2, ..., x_{10})$, $Sales(x_1, y_1, z_1, d_1)$ et $Produit(y_1$
$,y_2, ..., , y_{10})$.
Différentes instances de cette base de données sont considérées par les auteurs qui font varier successivement la cardinalité des domaines des motifs de 10, 40 et 80 pour un seuil de support de 0.05 et le nombre moyen de conditions de sélection à 4.
Ainsi, ils obtiennent les instances I10-1-5, I40-1-5, I80-1-5 pour lesquelles les cardinalités des tables $Cust$, $Sales$ et $Prod$ sont respectivement 2000, 6000 et 54000. Pour le deuxième groupe, les auteurs font varier le nombre moyen de conditions de sélection et ils obtiennent les instances TM3-1-5 et TM5-1-5.
Enfin, les auteurs considèrent les instances I10-3 et I10-5 pour voir l'influence du seuil de support pour une même instance.

Soit C = $\langle R, B, \Sigma \rangle$ un contexte et S une condition de sélection, rappelons que pour la sélection, nous pouvons avoir deux cas :

- Si $S \in \Sigma^*$, alors on sait que $freq_\alpha(\sigma_S^b(C))$ est un sous-ensemble de $freq_\alpha(C)$. L'ensemble des motifs fréquents de $freq_\alpha(\sigma_S^b(C))$ est généré à partir de la frontière positive et l'évaluation des supports des motifs se fait en une seule passe sur la base de données. C'est ce que les auteurs appellent évaluation en une seule passe.
- Si $S \notin \Sigma^*$, on ne sait pas si les motifs sont fréquents. Il faut les générer et effectuer le calcul des supports niveau par niveau. C'est l'évaluation en N passes, où N est le niveau maximal du treillis.

Les tests pour les sélections ont été faits comme suit : les auteurs considèrent d'abord le contexte C suivant :

- Requête de référence : $AllSales = \pi_{x_1}(Sales)$;
- Requête de base : $Cust \bowtie Sales \bowtie prod$;
- $\Sigma = \{x_1 = *, x_2 = *, x_4 = *, y_1 = *, y_2 = *, y_3 = *\}$, où $x = *$ veut dire que x peut prendre toutes les valeurs qui sont dans son domaine.

Ceci constitue une première requête d'extraction que les auteurs calculent et stockent. Les auteurs calculent ensuite l'ensemble des requêtes de $\sigma_S^b(C)$, où S est la condition de sélection vide. Ce calcul incrémental qui utilise les résultats de la première extraction fera l'objet d'une comparaison avec le premier calcul. Soulignons qu'avec la condition de sélection vide, l'évaluation peut se faire en une passe. Les auteurs ont également calculé les gains en temps en effectuant une évaluation en N passes pour tester ce que seraient les gains si $S \notin \Sigma^*$.

Les auteurs obtiennent ainsi des gains de temps variant de 45% à 88% selon les instances considérées. Ce gain de temps s'explique par une modification substantielle des temps d'évaluation du cas non incrémental au cas incrémental.
On peut noter également, que les gains augmentent d'une instance à l'autre proportionnellement à la cardinalité des domaines des attributs. Cela s'explique par la diminution du nombre de fréquents.

Pour le deuxième groupe (TM-3, TM-5), on peut faire sensiblement le même raisonnement. Les gains importants en temps de calcul s'expliquent par le pourcentage important de candidats élagués qui est ici respectivement de 97.9% et 91.8%. Ici, on remarque que les gains diminuent lorsque la taille moyenne des motifs augmente. En effet, plus le motif est de taille importante plus il y a de motifs fréquents, puisque tous ses sous-motifs sont fréquents. On remarque ici que le nombre de fréquents passe de 1917 pour TM3-1-5 à 7333 pour TM5-1-5. Finalement, si on compare les instances I-10-5 et I-10-3, on note une réduction sensible du gain en temps de calcul. Cela est normal puisqu'il y a une augmentation importante du nombre de fréquents, puisqu'il y a moins de cas d'élagage.
Pour la jointure, les auteurs considèrent les contextes $\mathcal{C}_1$, $\mathcal{C}_2$ et $\mathcal{C}_{12} = \mathcal{C}_1 \bowtie \mathcal{C}_2$ définis comme suit :

- $\mathcal{C}_1$:
  - Requête de référence : $AllCust = \pi_{x_1}(Sales)$;
  - Requête de base : $Cust \bowtie Sales$;
  - $\Sigma = \{y_1 = *, x_2 = *, x_3 = *, x_4 = *\}$
- $\mathcal{C}_2$:
  - Requête de référence : $AllCust = \pi_{x_1}(Sales)$;
  - Requête de base : $Sales \bowtie Prod$;
  - $\Sigma = \{y_1 = *, y_2 = *, y_3 = *\}$

Pour la jointure, les tests sont effectués de la même façon que pour la sélection. des différentes instances du premier et du deuxième groupe et le tableau des instances du quatrième groupe sont les mêmes que ceux pour la sélection .

Dans le cas de la jointure, les auteurs obtiennent également des gains de temps par rapport au cas non incrémental, et on peut raisonner de manière générale de la même façon que pour le cas de la sélection. Par exemple, pour l'instance I10, la diminution du seuil minimal de support de 5 à 3 entraîne une réduction du gain en temps de calcul de 8.2% à

1.8%, et l'on constate que le pourcentage de candidats élagués diminue aussi. Cependant, les gains obtenus dans le cas de la jointure sont moins importants que dans le cas de la sélection. On trouve par exemple des gains de 8.2%, 29.6% et 33.2% respectivement pour les instances I10-1-5, I40-1-5 et I80-1-5 alors qu'ils étaient de 56.0%, 81.0% et 89.8%.

Cette approche améliore sensiblement l'approche non incrémentale. Cependant, il faut noter comme nous l'avons dit précédemment que cette approche ne considère qu'un ensemble restreint de l'espace global. De ce fait, de nombreuses exécutions de l'algorithme seraient nécessaires pour envisager la découverte de l'ensemble des requêtes fréquentes.

2.7 Conclusion

Dans cette partie, nous avons fait l' état de l'art sur la découverte de motifs intéressants dans le bases de données. Nous avons tout d'abord présenté le formalisme général de recherche de motifs dans un ensemble de données proposé par Mannila et Toivonen ([89]), puis nous avons montré que ce formalisme s'applique aux problèmes de la recherche d'itemsets fréquents dans une table transactionnelle communément appelé problème du panier de la ménagère, mais également au problème de la recherche de motifs dans les bases de données relationnelles.

Le problème de la recherche de requêtes fréquentes dans les bases de données relationnelles qui est l'objet de cette thèse a ensuite été présenté. Nous avons montré les limites des méthodes dédiées au problème de la recherche d'itemsets fréquents dans les tables transactionnelles quand il s'agit de rechercher des requêtes fréquentes dans les bases de données relationnelle.

Afin de surmonter ces limites, quelques approches dédiées à la recherche de requêtes fréquentes dans les bases de données relationnelles ont été proposées par la communauté scientifique. Dans ce cadre, nous avons présenté les approches `WARMR` et `IncMiner` ([31, 36]), mais comme nous avons pu le constater, celles-ci ne considèrent qu'un ensemble restreint de l'espace global à cause de la taille importante de celui-ci, ce qui laisse de coté beaucoup de motifs potentiellement intéressants.Nous avons également présenté les seules approches existantes à notre connaissance qui permettent de rechercher l'ensemble des requêtes fréquentes dans une base de données relationnelle à savoir, `WARMER` ([54]) et `Conqueror` ([56]) . Cependant, ces deux approches ne prennent pas en compte les dépendances fonctionnelles et les dépendances d'inclusion présentes dans les bases de données relationnelles. De plus, l'évaluation des requêtes candidates au niveau de ces méthodes n'est pas optimale car utilisant des requêtes SQL. En effet, les E/S avec la base de données sont très couteuses, ce qui altère les performances de ces deux approches.
L'approche que nous allons présenter, et qui sera notre contribution dans ce cadre, est basée sur la prise en compte des dépendances fonctionnelles et des dépendances d'inclusion présentes dans les bases de données relationnelles pour rechercher efficacement les requêtes fréquentes. De plus, l'approche que nous présenterons contrairement aux approches précitées, qui utilisent le formalisme logique ou le langage SQL pour évaluer les supports des requêtes, nous proposerons, une manière efficiente d'évaluation des supports des requêtes basée sur l'utilisation d'une table auxilliaire AUX.

Il est également à noter qu'une approche, baptisée `Conqueror`$^+$, qui utilise les idées

de notre contribution et qui améliore `Conqueror` a été récemment proposée dans [99]. En effet, `Conqueror`$^+$ prend en compte les dépendances fonctionnelles et les dépendances d'inclusion présentes dans la base de données, pour optimiser l'extraction des requêtes fréquentes. Cette approche permet également la découverte de nouvelles dépendances fonctionnelles, de plus, les dépendances fonctionnelles découvertes sont utilisées pour optimiser les extractions futures.
Comme `Conqueror`$^+$ utilise les idées de notre contribution, sa description sera faite à la suite de notre contribution dans le chapitre suivant.

3
CONTRIBUTION

3.1 Introduction

Dans le chapitre précédent, nous avons pu noter que le problème de la recherche de requêtes fréquentes dans les bases de données relationnelles a attiré l'attention de la communauté scientifique durant ces dernières années.

Notre contribution dans ce cadre consiste à proposer un algorithme par niveau qui utilise une relation de pré-ordre sur l'ensemble des requêtes considérées, de la même manière que Apriori [8] en ce qui concerne les itemsets fréquents.

Cette relation de pré-ordre sera définie grâce aux dépendances fonctionnelles et dépendances d'inclusion présentes dans les bases de données relationnelles.

De plus, les classes d'équivalence induite par cette relation de pré-ordre, nous permet d'évaluer une requête par classe d'équivalence, car comme nous le verrons dans la suite, deux requêtes appartenant à une même classe d'équivalence ont le même support. Nous verrons également, que la relation d'équivalence que nous définissons est plus générale que l'inclusion diagonale proposée par [56], ce qui nous permet d'avoir un algorithme plus efficace.

En ce qui concerne l'évaluation des supports des classes de requêtes candidates, au lieu d'utiliser des requêtes SQL, nous présenterons une méthode qui utilise une table auxiliaire (AUX), qui permet d'évaluer efficacement le support des requêtes candidates.

De plus, nous nous focaliserons sur les bases de données de type schéma étoile pour lesquelles notre approche s'applique aisément.

Ce chapitre est constitué de sept sections. La première section est l'introduction. La section 3.2 présentera les notations et les définitions utilisées. La section 3.3 présentera notre approche. La section 3.4 quant à elle, présentera nos algorithmes. Dans la section 3.5, nous montrerons que le nombre de passes sur la base de donnée est linéaire par rapport à la taille de l'univers. La section 3.6 présentera une approche récente baptisés `Conqueror`$^+$ qui utilise les idées de notre approcheFinalement, la dernière section fera la conclusion de ce chapitre.

3.2 Modèle Formel

3.2.1 Introduction

Nous utilisons les notations classiques utilisées pour les bases de données relationnelles comme dans [115]. Afin de simplifier la notation, un tuple sera noté par le caractére minuscule et un schéma par le caractére majuscule correpondant. De plus, soit un tuple t sur X, pour tout sous-ensemble Y de X, $t.Y$ représente la restriction de t sur Y. L'univers *(l'ensemble de tous les attributs)* sera noté U.

Définition 3.1. *Nous rappelons les définitions de base concernant les dépendances fonctionnelles du modèle relationnel ([115]). Soient r une relation définie sur un schéma R et X, Y deux schémas de R. On dit que r satisfait une dépendance fonctionnelle de X vers Y si et seulement si : pour tout couple de tuples t, t' de r, $t.X = t'.X \Rightarrow t.Y = t'.Y$.*
Soit un ensemble $\mathcal{F}$ de dépendances fonctionnels (DF)*, nous notons par $\mathcal{F}^+$: l'ensemble des dépendances fonctionnelles pouvant être obtenues à partir de $\mathcal{F}$ par l'utilisation des axiomes de Armstrong ([11]).*
Soit un ensemble d'attributs X, la fermeture de X par rapport à $\mathcal{F}$, notée X^+, est l'ensemble des attributs A de R tels que $X \rightarrow A$ appartient à $\mathcal{F}^+$.
De plus, nous considérons pour les besoins de notre approche, que le schéma vide est noté $\emptyset$ et nous supposons que $dom(\emptyset)$ contient un seul élément, appelé tuple vide*, noté $\top$.*
Nous supposons également que :

- *Pour tout schéma X (vide ou non), chaque table r satisfait la dépendance fonctionnelle $X \rightarrow \emptyset$.*
- *Pour tout ensemble de dépendances fonctionnelles $\mathcal{F}$, $\emptyset^+ = \emptyset$.*

Remarque 3.1. *Nous verrons dans la suite de ce chapitre que l'extension des dépendances fonctionnelles au schéma vide est nécessaire dans notre approche.*

Dans la suite de ce chapitre, pour simplifier la notation, l'union de X et Y sera notée XY. De manière similaire, si A est un attribut, $X \cup \{A\}$ sera noté XA.

3.2.2 Requêtes

Nous considérons, une base données Δ contenant n tables *(ou relations)* $\{r_1, r_2, \ldots r_n\}$ définies sur des schémas notés $\{R_1, R_2, \ldots R_n\}$, respectivement.
De plus, nous supposons que chaque *relation* r_i $(i = 1, \ldots, n)$ est associée à un ensemble de dépendances fonctionnelles sur R_i noté $\mathcal{F}_i$; $\mathcal{F}$ représente l'union de tous les ensembles de dépendances fonctionnelles $\mathcal{F}_i$, pour $i = 1, \ldots, n$. Nous supposons également que les *relations* vérifient un ensemble de *dépendances d'inclusion* $\mathcal{I}$ qui sont de la forme $\pi_X(r_i) \subseteq \pi_X(r_j)$, où r_i et r_j sont dans Δ avec $X = R_i \cap R_j$.
En considérant les notations précédentes, nous définissons dans ce qui suit, de manière récursive, la notion de *key-foreign key join*.

Définition 3.2. *Soient deux relations r et s, respectivement définies sur les ensembles d'attributs R et S, nous notons $r \lhd s$ si nous avons simultanément les deux propriétés suivantes :*

Cust	*Cid*	*Cname*	*Caddr*
	c_1	John	Paris
	c_2	Mary	Paris
	c_3	Jane	Paris
	c_4	Anne	Tours

Prod	*Pid*	*Ptype*
	p_1	milk
	p_2	beer

Sales	*Cid*	*Pid*	*Qty*
	c_1	p_1	10
	c_2	p_2	5
	c_2	p_1	1
	c_1	p_2	10

$\mathcal{F}$: $Cid \rightarrow CnameCaddr$
$Pid \rightarrow Ptype$
$CidPid \rightarrow Qty$

$\mathcal{I}$: $\pi_{Cid}(Sales) \subseteq \pi_{Cid}(Cust)$
$\pi_{Pid}(Sales) \subseteq \pi_{Pid}(Prod)$

FIGURE 3.1 – Base de données de référence

1. *r satisfait la dépendance fonctionnelle $R \cap S \rightarrow R$, et*
2. *$\pi_{R \cap S}(s) \subseteq \pi_{R \cap S}(r)$.*

Soit $J = r_{i_1} \bowtie \ldots \bowtie r_{i_k}$ la jointure des relations Δ. La table J est appelé key-foreign key join*, ou* kfk-join *tout simplement, si la table J est égale à la jointure $J_1 \bowtie J_2$, où J_1 et J_2 sont soit des relations dans Δ ou des* kfk-joins *pour lesquelles la relation $J_1 \lhd J_2$ est vérifiée.*

Nous pouvons remarquer qu'une relation r est également un *kfk-join*, en effet, soit R le schéma de r nous avons :

1. $r \cap r \rightarrow r$ et
2. $\pi_{R \cap R}(r) \subseteq \pi_{R \cap R}(r)$.

En considérant l'Exemple 3.1, $Cust \bowtie Sales$ est un *kfk-join*, car $Cust$ et $Sales$ sont des relations de Δ, $Cid = D_1 \cap F \rightarrow Cust$ (D_1 et F étant respectivement les schémas de $Cust$ et $Sales$), et la dépendance d'inclusion $\pi_{Cid}(Sales) \subseteq \pi_{Cid}(Cust)$ est dans $\mathcal{I}$.
De manière similaire, $J = Cust \bowtie Prod \bowtie Sales$ est également un *kfk-join*, car $Prod$ et $Cust \bowtie Sales$ sont des *kfk-join* et nous avons $Prod \lhd (Cust \bowtie Sales)$.
En général, il est facile de montrer que les *kfk-joins* satisfont les propriétés énoncées dans le lemme suivant, où $|q|$ représente la cardinalité de la réponse à la requête q.

Lemme 3.1. *Soient r et s deux tables relationnelles définies respectivement sur R et S, telles que $r \lhd s$. Nous avons :*
- *$\pi_S(r \bowtie s) = s$ et ,*
- *$|r \bowtie s| = |s|$.*

Démonstration. L'inclusion $\pi_S(r \bowtie s) \subseteq s$ est triviale, nous allons juste montrer l'inclusion dans l'autre sens c'est à dire $s \subseteq \pi_S(r \bowtie s)$. Soit $t' \in s$, alors $t'.(R \cap S) \in \pi_{R \cap S}(s)$, et, comme $\pi_{R \cap S}(s) \subseteq \pi_{R \cap S}(r)$, il existe $t \in r$ tel que $t.(R \cap S) = t'.(R \cap S)$, et comme $R \cap S \rightarrow R$, $R \cap S$ est une clé de R, d'où t est unique.

Donc $tt' \in r \bowtie s$, soit $t' \in \pi_S(r \bowtie s)$ et $|r \bowtie s| = |s|$ comme tout tuple t de s est joint de manière unique à un tuple unique de r. Ce qu'il fallait démontrer. □

Nous notons, que le calcul de tous les *kfk-joins* est difficile dans le cas général, car le problème du test d'inclusion entre requêtes conjonctives, vérifiant un ensemble de dépendances fonctionnelles et d'inclusion, est de complexité NP ([71]). Ce problème ne sera pas traité car il dépasse le cadre de cette thèse. Ainsi, nous allons porter notre attention dans cette thèse au cas particulier des bases de données ayant un schéma étoile, car pour ce type de schéma de base de données, la caractérisation des *kfk-joins* est simple.
La Définition 3.3 donne la notion de requête conjonctive définie sur une relation r telle que définie dans cette contribution.

Définition 3.3. *Une* conjonction de conditions de sélection*, ou une condition de sélection pour simplifier, est une égalité de la forme $Y = y$ où Y schéma éventuellement vide et y un tuple de $dom(Y)$. Soit $S = (Y = y)$ une condition de sélection, un tuple t sur R satisfait S si $Y \subseteq R$:*
- *soit $Y = \emptyset$ et $y = \top$;*
- *soit $Y \neq \emptyset$ et $t.Y = y$.*

Nous noterons par $\mathcal{Q}$, l'ensemble des requêtes de la forme $q = \pi_X(\sigma_{Y=y}(r))$ où r est un kfk-join *et $XY \subseteq R$ (R représentant le schéma de r).*
Soit $q = \pi_X(\sigma_{Y=y}(r))$ dans $\mathcal{Q}$, la réponse à q dans Δ *est définie de manière usuelle si $X \neq \emptyset$. Si $X = \emptyset$:*
- *$\pi_\emptyset(\sigma_{Y=y}(r)) = \{\top\}$, si $y \in \pi_Y(r)$,*
- *$\pi_\emptyset(\sigma_{Y=y}(r)) = \pi_\emptyset(\emptyset) = \emptyset$, sinon.*

Pour simplifier la notation, la requête $\pi_X(\sigma_{Y=y}(r))$ de $\mathcal{Q}$ sera notée simplement $\pi_X\sigma_y(r)$.

Par exemple, la requête $\pi_{Cid}(\sigma_{Caddr=\texttt{Paris}}(Cust))$ sera notée simplement $\pi_{Cid}\sigma_{\texttt{Paris}}(Cust)$.
Nous notons, si r est un *kfk-join* sur le schéma R, les requêtes telles que $\sigma_y(r)$ ou $\pi_X(r)$ sont également des requêtes de $\mathcal{Q}$.
En effet, d'une part, il est clair que la requête $\sigma_y(r)$ peut être écrite sous la forme $\pi_R\sigma_y(r)$ et, d'autre part, comme tout tuple t de la relation r satisfait la condition de sélection ($\emptyset = \top$), les requêtes $\pi_X(r)$ peuvent être écrites sous la forme $\pi_X\sigma_\top(r)$.
En conséquence, toute relation r peut être écrite sous la forme $\pi_R\sigma_\top(r)$.
Ainsi, pour pouvoir écrire les relations sous forme de requêtes, il faut que les conditions de sélection sur les schémas vides soient définies.

L'exemple suivant illustre la Définition 3.3.

Exemple 3.1. *En considérant l'Exemple 3.1, il est facile de voir que les requêtes $q_1 = \pi_{Cid}\sigma_{\texttt{Paris}}(Cust)$ et $q_2 = \pi_{Cid}\sigma_{\texttt{Paris beer}}(Cust \bowtie Prod \bowtie Sales)$ sont dans $\mathcal{Q}$.*
Les réponses aux requêtes q_1 et q_2 sont respectivement $\{\texttt{c}_1, \texttt{c}_2, \texttt{c}_3\}$ et $\{\texttt{c}_1, \texttt{c}_2\}$.
D'autre part, $\pi_{Cid}(Cust)$ et $\pi_{Cid}(Cust \bowtie Sales)$ sont des requêtes dans $\mathcal{Q}$ qui peuvent être représentées respectivement par $\pi_{Cid}\sigma_\top(Cust)$ et $\pi_{Cid}\sigma_\top(Cust \bowtie Sales)$, et les réponses sont respectivement $\{\texttt{c}_1, \texttt{c}_2, \texttt{c}_3, \texttt{c}_4\}$ et $\{\texttt{c}_1, \texttt{c}_2\}$.
Considérons maintenant, les deux requêtes de $\mathcal{Q}$:
- *$\pi_{Pid}\sigma_{\texttt{beer 15}}(Prod \bowtie Sales)$ et*
- *$\pi_\emptyset\sigma_{\texttt{beer 15}}(Prod \bowtie Sales)$,*

comme `beer 15` *n'est pas dans* $\pi_{Ptype\,Qty}(Prod \bowtie Sales)$*, les réponses à ces deux requêtes sont vides.*
Notons, cependant, que le remplacement de `15` *par* `5` *dans ces deux requêtes donne respectivement* $\{p_2\}$ *et* $\{\top\}$ *comme réponse, car* `beer 5` *est dans* $\pi_{Ptype\,Qty}(Prod \bowtie Sales)$.

Le support ou la fréquence d'une requête est définie de la manière suivante :

Définition 3.4. *Soit une requête q, nous notons par $ans(q)$ la réponse de la requête q. Pour toute requête q de $\mathcal{Q}$, le* support de q dans Δ*, noté $sup(q)$, est la cardinalité de la réponse à q. Soit un seuil de support min-sup, une requête q sera dite* fréquente*, si $sup(q) \geq$ min-sup.*

En reprenant les requêtes q_1 et q_2 de l'Exemple 3.1, nous avons $sup(q_1) = 3$ et $sup(q_2) = 2$. Ainsi, si *min-sup* $= 3$, q_1 est fréquente et q_2 est *non fréquente*.
Nous noterons que, pour toute requête $q = \pi_X \sigma_y(r)$ dans $\mathcal{Q}$ telle que $y \notin \pi_Y(r)$, nous avons $sup(q) = 0$. De plus, si $X \subseteq Y$, alors $sup(q)$ est égal à 1 si $y \in \pi_Y(r)$ ou 0 sinon. D'autre part, il est clair que si $r \neq \emptyset$ et $X = Y = \emptyset$, alors $sup(q) = 1$.

Les résultats suivants montrent qu'il est possible de comparer les supports des requêtes en utilisant les dépendances fonctionnelles et les dépendances d'inclusion. Dans ce qui suit, $r_Y(x)$ représentera la restriction du tuple de r associé à x sur Y relativement à la dépendances fonctionnelle $X \rightarrow Y$.

Lemme 3.2. *Soient r une relation définie sur le schéma R, et X et Y deux sous-schémas non vides de R. Si r satisfait $X \rightarrow Y$ alors :*

1. $sup(\pi_X(r)) \geq sup(\pi_Y(r))$ *et*
2. $sup(\sigma_{X=x}(r)) \leq sup(\sigma_{Y=r_Y(x)}(r))$.

Démonstration.
1. Comme r vérifie $X \rightarrow Y$, il existe une fonction surjective entre $ans(\pi_X(r))$ et $ans(\pi_Y(r))$. Ainsi, $sup(\pi_X(r)) \geq sup(\pi_Y(r))$
2. Pour tout tuple t de $ans(\sigma_{X=x}(r))$, $t.X = x$. Ainsi, $t.Y = r_Y(x)$, ce qui montre que $t \in ans(\sigma_{Y=r_Y(x)}(r))$. D'où finalement, $sup(\sigma_{X=x}(r)) \leq sup(\sigma_Y = r_Y(x)(r))$.

□

Dans ce qui suit, nous allons tout d'abord donner la définition d'une base de données ayant schéma étoile puis nous présenterons notre contribution à la recherche de l'ensemble des requêtes fréquentes dans une base de données définie un schéma étoile.

3.3 Recherche de requêtes fréquentes dans les bases de données définies sur un schéma étoile

3.3.1 Introduction

Nous rappelons, qu'une base de données définie sur un schéma étoile ([60]) est une base de données composée d'une table φ ayant un schéma F donné, appelée *table de faits*, et un ensemble de tables $\delta_1, \ldots, \delta_N$ ayant respectivement pour schéma $D_1, \ldots, D_N$, appelées *tables de dimension*, telles que :

1. Si $K_1, \ldots, K_N$ sont les clés (primaires) de $\delta_1, \ldots, \delta_N$ respectivement, et si K est l'union de ces clés (*i.e.,* $K = K_1 \ldots K_N$), alors K est la clé de φ ;
2. Pour chaque $i = 1, \ldots, N$, $\pi_{K_i}(\varphi) \subseteq \pi_{K_i}(\delta_i)$ (K_i étant une clé étrangère de la table φ).

L'ensemble d'attributs $M = F \setminus K$ est appelé *mesure* du schéma étoile. Avec ces notations, nous avons $U = D_1 \ldots D_N F$, les dépendances fonctionnelles $\mathcal{F}$ et les dépendances d'inclusion $\mathcal{I}$ s'écrivent respectivement avec les deux ensembles :

- $\mathcal{F} = \{K_1 \to D_1, \ldots, K_N \to D_N, K \to M\}$
- $\mathcal{I} = \{\pi_{K_1}(\varphi) \subseteq \pi_{K_1}(\delta_1), \ldots, \pi_{K_N}(\varphi) \subseteq \pi_{K_N}(\delta_N)\}$.

En reprenant notre base de données exemple représentée par la Figure 3.1, nous remarquons que la base de données Δ est définie sur un schéma étoile où $Sales$ est la table de fait, et $Cust$ et $Prod$ sont les tables de dimensions, et l'ensemble de dépendances fonctionnelles. $\mathcal{F}$ est alors $\{Cid \to CidCnameCaddr, Pid \to PidPtype, CidPid \to CidPidQty\}$ et $\mathcal{I} = \{\pi_{Cid}(Sales) \subseteq \pi_{Cid}(Cust), \pi_{Pid}(Sales) \subseteq \pi_{Pid}(Prod)\}$.
La proposition suivante permet de caractériser les *kfk-joins* dans le cas des bases de données ayant un schéma étoile.

Proposition 3.1. *Si Δ est une base de données ayant un schéma étoile, alors une table J est un* kfk-join *si et seulement si J est une table de Δ, ou J est une jointure contenant la table de faits φ.*

Démonstration. Si J est une unique relation, le résultat vient de notre remarque précédente, et si J est une jointure contenant la table φ, puisque pour tout $i \in \{1, \ldots, N\}$, $D_i \lhd \varphi$.
Si J est une jointure autre que celles spécifiées dans la proposition, alors J n'est pas réduite à une seule relation de Δ et J ne contient pas φ non plus.
Ainsi, J est la jointure de deux ou de plusieurs tables de dimension, et puisque pour tout $i, j \in \{1, \ldots, N\}^2$ et $i \neq j$, $D_i \cap D_j = \emptyset$, la relation $D_i \lhd D_j$ n'est pas vérifiée. Ainsi, nous en déduisons que J ne peut pas être un *kfk-join*. Ce qui complète la preuve. □

La conséquence de la Proposition 3.1 est la suivante : les requêtes de $\mathcal{Q}$ sont de la forme $\pi_X \sigma_y(r)$ où r est soit une table de Δ ou une jointure des tables de Δ contenant φ.

3.3.2 Relation de pré-ordre sur les requêtes d'une base de données ayant un schéma étoile

La définition suivante donne la définition de la relation de pré-ordre proposée dans cette contribution.

Définition 3.5. *Soient deux tables relationnelles r_1, r_2 et deux requêtes $q_1 = \pi_{X_1}\sigma_{y_1}(r_1)$ et $q_2 = \pi_{X_2}\sigma_{y_2}(r_2)$ de $\mathcal{Q}$, q_1 est dite plus générale que q_2, que l'on note $q_1 \preceq q_2$, si l'on a l'une des propriétés suivantes :*

1. $y_2 \notin \pi_{Y_2}(r_2)$
2. $y_2 \in \pi_{Y_2}(r_2)$, $y_1 \in \pi_{Y_1}(r_1)$ *et* $Y_1 \to X_1 \notin \mathcal{F}^+$.
3. q_1 *et* q_2 *sont telles que :*

- $r_2 \lhd r_1$
- $X_2Y_1 \rightarrow X_1 \in \mathcal{F}^+$,
- $Y_1 \rightarrow Y_2 \in \mathcal{F}^+$,
- $y_1y_2 \in \pi_{Y_1Y_2}(r_1 \bowtie r_2)$.

Afin de démontrer l'anti-monotonie de la relation de pré-ordre de la Définition 3.5 démontrons tout d'abord la proposition suivante :

Proposition 3.2. *Soient une relation r définie sur un schéma R, et deux requêtes $q_1 = \pi_{X_1}\sigma_{y_1}(r)$ et $q_2 = \pi_{X_2}\sigma_{y_2}(r)$ sur r vérifiant l'une des trois items suivants :*

1. $y_1 \notin \pi_{Y_1}(r)$;
2. $y_1 \in \pi_{Y_1}(r)$, $y_2 \in \pi_{Y_1}(r)$ *et* $Y_2 \rightarrow X_2 \in \mathcal{F}^+$;
3. $y_1 \in \pi_{Y_1}(r)$, $y_2 \in \pi_{Y_1}(r)$ *et* $Y_2 \rightarrow X_2 \notin \mathcal{F}^+$;
   - $X_1Y_2 \rightarrow X_2 \in \mathcal{F}^+$;
   - $Y_2 \rightarrow Y_1 \in \mathcal{F}^+$;
   - $y_1y_2 \in \pi_{Y_1Y_2}(r)$.

alors nous avons : $sup(q_2) \leq sup(q_1)$.

Démonstration.
1. Soit $y_2 \notin \pi_{Y_2}(r)$, dans ce cas, il est évident que $sup(q_2) = 0$, ainsi, $sup(q_2) \leq sup(q_1)$.
2. Soient $y_2 \in \pi_{Y_2}(r)$ et $y_1 \in \pi_{Y_1}(r)$.
   Dans le cas où $Y_1 \rightarrow X_1$, il est évident que $sup(q_1) = 1$, $y_1y_2 \in \pi_{Y_1Y_2}(r)$ et comme $y_2 \in \pi_{Y_2}(r)$, $sup(q_2) \geq 1$ d'où $sup(q_2) \leq sup(q_1)$.
3. Soient $y_1 \in \pi_{Y_1}(r)$, $y_2 \in \pi_{Y_2}(r)$, $Y_1 \rightarrow X_1 \notin \mathcal{F}^+$.
   - si $y_1 = \top$, nous avons $Y_1 = \emptyset$, et comme $Y_1 \rightarrow Y_2 \in \mathcal{F}^+$ et $\emptyset^+ = \emptyset$, nous en déduisons que $Y_2 = \emptyset$. Ainsi, $y_1 = \top$, comme $\sigma_\top(r) = r$ et $X_2 \rightarrow X_1$ (puisque $Y_1 = \emptyset$), grâce au Lemme 3.2.(1), comme $y_1y_2 \in \pi_{Y_1Y_2}(r)$, nous déduisons que $sup(q_2) \leq sup(q_1)$.
   - si $y_1 \neq \top$, $ans(\pi_R\sigma_{y_1}(r))$ satisfait $X_2 \rightarrow Y_1$ (car pour tout tuple t de $\sigma_{y_1}(r)$, $t.Y_1 = y_1$), et puisque $X_2Y_1 \rightarrow X_1$ est satisfait dans r, $ans(\pi_R\sigma_{y_1}(r))$ satisfait également $X_2 \rightarrow X_1$ (comme $ans(\pi_R\sigma_{y_1}(r))$ satisfait $X_2 \rightarrow Y_1$). Et sachant que : $ans(\pi_{X_2}\sigma_{y_1}(r)) = \pi_{X_2}(ans(\pi_R\sigma_{y_1}(r)))$ et $ans(\pi_{X_1}\sigma_{y_1}(r)) = \pi_{X_1}(ans(\pi_R\sigma_{y_1}(r)))$, nous en déduisons que $sup(\pi_{X_1}\sigma_{y_1}(r)) \leq sup(\pi_{X_2}\sigma_{y_1}(r))$.
     Finalement, comme $Y_1 \rightarrow Y_2 \in \mathcal{F}(r)^+$, comme $y_1y_2 \in \pi_{Y_1Y_2}(r)$, grâce au Lemme 3.2.(2), nous en déduisons que $ans(\pi_{X_2}\sigma_{y_1}(r)) \leq ans(\pi_{X_2}\sigma_{y_2}(r))$, d'où finalement $sup(\pi_{X_2}\sigma_{y_2}(r)) \leq sup(\pi_{X_1}\sigma_{y_1}(r))$.

□

Le corollaire suivant basé sur la Proposition 3.2 permet d'établir l'anti-monotonie du support par rapport à la relation de pré-ordre $\preceq$ dans le cas de deux requêtes définies sur des tables relationnelles différentes vérifiant la Définition 3.5 :

Corollaire 3.1. *Soient deux tables relationnelles r_1 et r_2 et deux requêtes $q_1 = \pi_{X_1}\sigma_{y_1}(r_1)$ et $q_2 = \pi_{X_2}\sigma_{y_2}(r_2)$ de $\mathcal{Q}$, si $q_1 \preceq q_2$ alors $sup(q_2) \leq sup(q_1)$.*

Démonstration. Remarquons tout d'abord que $\pi_{X_1}\sigma_{y_1}(r_1 \bowtie r_2) \subseteq q_1$, puisque $\pi_{R_i}(r_1 \bowtie r_2) \subseteq r_i$ $(i = 1, 2)$ pour toute relation r_1 et r_2. Ainsi, $sup(\pi_{X_1}\sigma_{y_1}(r_1 \bowtie r_2)) \leq sup(q_1)$ (1). En utilisant le Lemme 3.1, comme nous avons $r_1 \lhd r_2$, nous avons aussi $q_2 = \pi_{X_2}\sigma_{y_2}(r_1 \bowtie r_2)$, ce qui entraine $sup(q_2) = sup(\pi_{X_2}\sigma_{y_2}(r_1 \bowtie r_2))$ (2). En appliquant, la Proposition 3.2 à $r_1 \bowtie r_2$, nous avons $sup(\pi_{X_2}\sigma_{y_2}(r_1 \bowtie r_2)) \leq sup(\pi_{X_1}\sigma_{y_1}(r_1 \bowtie r_2))$.
Ainsi, grâce à (1) et (2), nous déduisons $sup(q_2) \leq sup(q_1)$, ce qu'il fallait démontrer. □

L'exemple suivant illustre le Corollaire 3.1.

Exemple 3.2. *Considérons les requêtes de l'Exemple 3.1,*
$q_1 = \pi_{Cid}\sigma_{\texttt{Paris}}(Cust)$ *et* $q_2 = \pi_{Cid}\sigma_{\texttt{Paris beer}}(Cust \bowtie Prod \bowtie Sales)$*, nous avons :*
- $Cust \lhd (Cust \bowtie Prod \bowtie Sales)$,
- $Cid\, Caddr\, Ptype \rightarrow Cid \in \mathcal{F}^+$,
- $Caddr\, Ptype \rightarrow Caddr \in \mathcal{F}^+$, *et*
- $(\texttt{Paris beer}) \in \pi_{Caddr\, Ptype}(Cust \bowtie Prod \bowtie Sales)$.

Ainsi, en utilisant le Corollaire 3.1, nous avons $sup(q_2) \leq sup(q_1)$.

Il est à noter qu'il est impossible de définir une relation de pré-ordre entre les requêtes basée sur la relation binaire $\lhd$. Ceci est dû au fait que la relation $\lhd$, n'est pas transitive en général, comme le montre l'exemple suivant :

Exemple 3.3. *Soit* Δ *une base de données ayant trois relations* r_1, r_2 *et* r_3 *respectivement définies sur les schémas* AB, BC *et* ACD, *nous supposons les dépendances fonctionnelles suivantes :* $\mathcal{F} = \{B \rightarrow A, C \rightarrow B\}$ *et* $\mathcal{I} = \{\pi_B(r_2) \subseteq \pi_B(r_1), \pi_C(r_3) \subseteq \pi_C(r_2)\}$. *Pour* $q_1 = \pi_C\sigma_\top(r_3)$, $q_2 = \pi_B\sigma_\top(r_2)$ *et* $q_3 = \pi_A\sigma_\top(r_1)$, *les conditions de la Proposition 3.2 sont satisfaites par* q_1 *et* q_2, *ainsi que par* q_2 *et* q_3. *En effet, nous avons* $r_1 \lhd r_2$, $r_2 \lhd r_3$, *grâce aux dépendances fonctionnelles* $B \rightarrow A$ *et* $C \rightarrow B$.
Cependant, pour $r_1 = \{ab\}$, $r_2 = \{bc\}$ *et* $r_3 = \{a'cd\}$, $\mathcal{F}$ *et* $\mathcal{I}$ *sont vérifiés, cependant* $r_1 \lhd r_3$ *n'est pas vérifié car* $\pi_A(r_3) \not\subseteq \pi_A(r_1)$. *Ainsi, dans ce cas nous montrons que les conditions de la Proposition 3.2 ne sont pas satisfaites par* q_1 *et* q_3.

Cependant, lorsque l'on considère les bases de données ayant un schéma étoile nous montrerons dans la suite que la relation $\lhd$ devient transitive. C'est pourquoi, nous nous focaliserons sur les bases de données définie sur un schéma étoile.
Pour définir une relation d'ordre sur les requêtes d'une base de données ayant un schéma étoile, nous utilisons la *relation de pré-ordre* $\preceq$ de la Définition 3.5
Il faut noter que la relation ainsi définie dépend de l'instance de base de données considérée. En effet, décider que $q_1 \preceq q_2$, nécessite de tester si les tuples qui apparaissent dans q_2 et/ou dans q_1 apparaissent dans certaines relations de la base.
Cependant, comme nous le verrons dans la suite, ceci n'impacte pas dans le calcul des requêtes fréquentes.
En effet, dans l'algorithme que nous présenterons dans la suite, seuls les tuples effectivement présents dans les relations seront considérés dans les conditions de sélection des classes de requêtes. Par suite la suite, nous verrons que le cas (1) de la Définition 3.5 n'est pas explicitement considéré dans notre algorithme. De plus, la Définition 3.5 montre que les requêtes $q = \pi_X\sigma_y(r)$ telles que $y \notin \pi_Y(r)$ constituent les requêtes les plus spécifiques

de $\mathcal{Q}$. Ces requêtes ont un support égal à 0 (c'est à dire la valeur de support la plus petite que nous puissions avoir).
D'autre part, dans le cas (2) de la Définition 3.5, comme y est dans $\pi_Y(r)$, nous avons toujours : $sup(q) \geq 1$. De plus, en supposant que y_1 est dans $\pi_{Y_1}(r)$ et comme $Y_1 \rightarrow X_1$ est dans $\mathcal{F}^+$, la réponse à q_1 est réduite à un tuple unique et ainsi, nous avons alors $sup(q_1) \leq sup(q)$.
Nous notons également que, si $N > 1$, pour chaque table de dimension δ_i, les requêtes $\pi_{D_i}\sigma_\top(\delta_i)$ et $\pi_F\sigma_\top(\varphi)$, *i.e.*, δ_i et φ ne sont pas comparables en utilisant $\preceq$. En effet, comme la relation $\varphi \lhd \delta_i$ n'est pas vérifiée, la relation $\varphi \preceq \delta_i$ ne peut être vérifiée , et si $\delta_i \lhd \varphi$ est vérifié, $\delta_i \preceq \varphi$ ne peut pas être vérifié, car $D_i \rightarrow F$ n'est pas dans $\mathcal{F}^+$.
De plus, pour $i \neq j$, $\pi_{D_i}\sigma_\top(\delta_i)$ et $\pi_{D_j}\sigma_\top(\delta_j)$ sont également deux requêtes non comparables de $\mathcal{Q}$, car ni la relation $D_i \lhd D_j$ ni la relation $D_j \lhd D_i$ n'est vérifiée.
Il est facile de voir que la relation $\pi_{D_i}\sigma_\top(\delta_i) \preceq \pi_{K_i}\sigma_\top(\varphi)$ est vérifiée, tandis que la relation $\pi_{K_i}\sigma_\top(\varphi) \preceq \pi_{D_i}\sigma_\top(\delta_i)$ n'est pas vérifiée.
L'exemple suivant illustre d'autres cas de requêtes comparables par rapport à la Définition 3.5.

Exemple 3.4. *Reprenons notre base de données exemple de la Figure 3.1 et considérons une fois encore les requêtes* $q_1 = \pi_{Cid}\sigma_{\texttt{Paris}}(Cust)$ *et* $q_2 = \pi_{Cid}\sigma_{\texttt{Paris beer}}(Cust \bowtie Prod \bowtie Sales)$. *En considérant la Définition 3.5.3, nous avons :*

(*a*) $Cust \lhd (Cust \bowtie Prod \bowtie Sales)$

(*b*) $\texttt{Paris} \in \pi_{Caddr}(Cust)$,
$\texttt{Paris beer} \in \pi_{Caddr\ Ptype}\ (Cust \bowtie Prod \bowtie Sales)$,
$Caddr\ Ptype \rightarrow Cid \notin \mathcal{F}^+$

(*c*) $Cid\,Caddr\,Ptype \rightarrow Cid \in \mathcal{F}^+$

(*d*) $Caddr\,Ptype \rightarrow Caddr \in \mathcal{F}^+$

(*e*) $\texttt{Paris beer} \in \pi_{Caddr\ Ptype}(Cust \bowtie Prod \bowtie Sales)$.

Nous en déduisons alors que : $q_1 \preceq q_2$.
Considérons maintenant les requêtes $q_2' = \pi_{Cname}\sigma_{\texttt{c}_2\ \texttt{beer}}(Cust \bowtie Prod \bowtie Sales)$. *Nous avons* $q_1 \preceq q_2'$ *à cause de la Définition 3.5.2.*
En effet, $\texttt{Paris} \in \pi_{Caddr}(Cust)$, $\texttt{c}_2\,\texttt{beer} \in \pi_{Cid\,Ptype}(Cust \bowtie Prod \bowtie Sales)$ *et* $Cid\,Ptype \rightarrow Cname \in \mathcal{F}^+$. *Soit* $q_3 = \pi_{Cid\,Cname}\sigma_\top(Cust \bowtie Sales)$. *En utilisant les mêmes arguments que précédemment nous avons* $q_3 \preceq q_2$.
En considérant $q_3' = \pi_{Cid\,Cname\,Caddr}\sigma_\top(Cust \bowtie Prod \bowtie Sales)$, *et en appliquant la Définition 3.5.(3), il est facile de voir que nous avons* $q_3 \preceq q_3'$ *et* $q_3' \preceq q_3$. *Soit maintenant,* $q_4 = \pi_\emptyset\sigma_{\texttt{beer 15}}(Prod \bowtie Sales)$, *en utilisant la Définition 3.5.(1), nous avons* $q_1 \preceq q_4$, $q_2 \preceq q_4$ *et* $q_3 \preceq q_4$, *car* $\texttt{beer 15}$ *n'est pas dans* $\pi_{Ptype\,Qty}(Cust \bowtie Sales)$. *D'autre part, en considérant* $q_5 = \pi_\emptyset\sigma_{\texttt{beer 5}}(Prod \bowtie Sales)$, *nous montrons que* $q_1 \preceq q_5$, $q_2 \preceq q_5$ *et* $q_3 \preceq q_5$. *En effet, en utilisant la Définition 3.5.2,* $\texttt{Paris} \in \pi_{Caddr}(Cust)$, $\texttt{beer 5} \in \pi_{Ptype\,Qty}(Prod \bowtie Sales)$ *et la dépendance fonctionnelle* $Ptype\,Qty \rightarrow \emptyset$ *est dans* $\mathcal{F}^+$. *Notons également que, d'après la Définition 3.5.(1), nous avons également* $q_5 \preceq q_4$.

La proposition suivante montre que la relation $\preceq$ est une relation de pré-ordre (réflexive et transitive) sur les requêtes d'une base de données relationnelle $\mathcal{Q}$.

Proposition 3.3. *La relation $\preceq$ est une relation de pré-ordre sur $\mathcal{Q}$.*

Démonstration. Il est facile de voir que $\preceq$ est réflexive, de ce fait, nous allons seulement montrer sa transitivité.
Soient les requêtes $q = \pi_X \sigma_y(r)$, $q_1 = \pi_{X_1} \sigma_{y_1}(r_1)$ et $q_2 = \pi_{X_2} \sigma_{y_2}(r_2)$ telles que $q \preceq q_1$ et $q_1 \preceq q_2$, montrons que $q \preceq q_2$.
Si $y_2 \notin \pi_{Y_2}(r_2)$, nous avons $q \preceq q_2$ en considérant la Définition 3.5.(1) précédente.
Supposons que $y_2 \in \pi_{Y_2}(r_2)$, dans ce cas $y_1 \in \pi_{Y_1}(r_1)$ (du fait que $q_1 \preceq q_2$), ce qui implique que $y \in \pi_Y(r)$ (du fait que $q \preceq q_1$), grâce à Définition 3.5.(2), si $Y_2 \rightarrow X_2$ est dans $\mathcal{F}^+$, nous avons $q \preceq q_2$.
Considérons maintenant le cas où $y_2 \in \pi_{Y_2}(r_2)$, $y_1 \in \pi_{Y_1}(r_1)$, $y \in \pi_Y(r)$ et $Y_2 \rightarrow X_2 \notin \mathcal{F}^+$. Dans ce cas, $X_1 Y_2 \rightarrow X_2 \in \mathcal{F}^+$ et $Y_2 \rightarrow Y_1 \in \mathcal{F}^+$ sont dans $\mathcal{F}^+$. si $Y_1 \rightarrow X_1 \in \mathcal{F}^+$, nous montrons que $Y_2 \rightarrow X_2 \in \mathcal{F}^+$, ce qui est une contradiction. Ainsi, comme $q \preceq q_1$ et $y \in \pi_Y(r_1)$, $XY_1 \rightarrow X_1 \in \mathcal{F}^+$ et $Y_1 \rightarrow Y \in \mathcal{F}^+$. Nous en déduisons que $XY_2 \rightarrow X_2 \in \mathcal{F}^+$ et $Y_2 \rightarrow Y \in \mathcal{F}^+$.
Supposons maintenant que la relation $r \lhd r_2$ ne soit pas vérifiée : Dans ce cas, r et r_2 sont deux tables de dimension distinctes et, comme nous avons $q_1 \preceq q_2$, pour que la relation $r_1 \lhd r_2$ soit satisfaite, il faut nécessairement que $r_1 = r_2$, de manière similaire nous déduisons également que $r = r_1$ (comme $q \preceq q_1$), en conséquence $r = r_2$, ce qui implique que la relation $r \lhd r_2$ est vérifiée ce qui est une contradiction. Ainsi, nous avons bien $r \lhd r_2$.

De plus, comme $yy_1 \in \pi_{YY_1}(r \bowtie r_1)$ et $y_1 y_2 \in \pi_{Y_1 Y_2}(r_1 \bowtie r_2)$, et puisque $Y_2 \rightarrow Y_1 \in \mathcal{F}^+$ et $Y_1 \rightarrow Y_2 \in \mathcal{F}^+$, nous avons $yy_2 \in \pi_{YY_2}(r \bowtie r_1 \bowtie r_2)$. De plus comme $r_1 \lhd (r \bowtie r_2)$ car r_2 est soit la table de jointure soit une table de dimension et dans ce cas $r_2 = r_1 = r$, grâce au Lemme 3.1, nous avons alors $r \bowtie r_2 = \pi_{RR_2}(r \bowtie r_1 \bowtie r_2)$. Ainsi, $\pi_{RR_2}(r \bowtie r_1 \bowtie r_2) = \pi_{RR_2}(r \bowtie r_2)$ et $yy_2 \in \pi_{YY_2}(r \bowtie r_2)$. Ainsi, grâce à la Définition 3.5.(3), nous avons bien $q \preceq q_2$, ce qu'il fallait démontrer. □

De plus, la Proposition 3.1 montre l'anti-monotonie de $\preceq$. Nous rappelons, comme ça a été démontré dans [8], que l'anti-monotonie du support par rapport à la relation de pré-ordre est une importante propriété permettant de parcourir de manière efficace l'espace de recherche. Dans notre contexte, nous utilisons la Proposition 3.1 de la même manière. Étant donné un seuil de support *min-sup* et deux requêtes q et q_1 de $\mathcal{Q}$ telles que $q \preceq q_1$, nous avons la propriété suivante : si q n'est pas fréquente, alors q_1 n'est pas fréquente. Ainsi, il n'est pas nécessaire de calculer $sup(q_1)$ si q est non fréquente. D'autres optimisations sont également possible si on utilise les classes d'équivalence de requêtes au lieu de considérer individuellement chaque requête. Dans ce qui suit, nous allons caractériser les classes d'équivalence.

3.3.3 Classes d'équivalence de requêtes

En considérant $\preceq$, deux requêtes q et q_1 de $\mathcal{Q}$ seront dites *équivalentes*, noté $q \equiv q_1$, si et seulement si $q \preceq q_1$ et $q_1 \preceq q$.
Nous notons par $\mathcal{C}$ l'ensemble des classes d'équivalence modulo $\equiv$, la relation de pré-ordre $\preceq$ sur $\mathcal{Q}$ induit une relation d'ordre partiel sur $\mathcal{C}$ que nous noterons également $\preceq$.

Cet ordre est défini de la manière suivante : si $[q]$ représente la classe d'équivalence de q modulo $\equiv$, pour toutes classes de requêtes $[q]$ et $[q_1]$ de $\mathcal{C}$, $[q]$ est dite *plus générale que* $[q_1]$, notée $[q] \preceq [q_1]$, si $q \preceq q_1$.
Il est facile de montrer que $\preceq$ est une relation d'ordre partiel sur $\mathcal{C}$ indépendante du choix des représentants. C'est pourquoi, nous utilisons la même notation pour la relation de pré-ordre dans $\mathcal{Q}$ qui est associée à la relation de pré-ordre dans $\mathcal{C}$.
L'important corollaire suivant qui nous sera utile dans la suite de cette thèse, est une conséquence directe de la Proposition 3.2.

Corollaire 3.1. *Pour toutes requêtes q et q_1 de $\mathcal{Q}$, si $q \equiv q_1$ alors $sup(q) = sup(q_1)$.*

En reprenant l'Example 3.4, nous pouvons voir que les requêtes $q_3 = \pi_{Cid\,Cname}\sigma_{\top}(Cust \bowtie Sales)$ et $q'_3 = \pi_{Cid\,Cname\,Caddr}\sigma_{\top}(Cust \bowtie Prod \bowtie Sales)$ sont telles que $q_3 \preceq q'_3$ et $q'_3 \preceq q_3$. Nous en déduisons que ces deux requêtes sont équivalentes, et donc le Corollaire 3.1 ci-dessous montre que ces requêtes ont le même support.
En utilisant le Corollaire 3.1, soit une classe $[q]$ de $\mathcal{C}$, nous notons par $sup([q])$ le support de $[q]$, *i.e.,* le support commun à toutes les requêtes de $[q]$. De la même manière que dans le cas des requêtes prises séparément, soit un seuil de support *min-sup*, une classe $[q]$ de $\mathcal{C}$ est dite *fréquente* si $sup([q]) \geq$ *min-sup*.
La proposition suivante découle de la Proposition 3.1 et du Corollaire 3.1.

Proposition 3.4. *Pour toutes classes de requêtes $[q]$ et $[q_1]$ de $\mathcal{C}$, si $[q] \preceq [q_1]$ alors $sup([q_1]) \leq sup([q])$.*

L'impact du Corollaire 3.1 et la Proposition 3.4 dans le calcul des requêtes fréquentes est le suivant :

1. Pour chaque classe d'équivalence seulement *un calcul* du support est nécessaire. Nous ne considérons pas individuellement les requêtes de $\mathcal{Q}$, mais plutôt les classes d'équivalence de $\mathcal{C}$.
2. Les classes fréquentes peuvent être calculées en utilisant un algorithme par niveau grâce à la propriété d'anti-monotonie.

Cependant, le fait de considérer les classes d'équivalence plutôt que les requêtes prises individuellement est une stratégie payante :
- si les classes d'équivalence peuvent être caractérisées facilement,
- et si l'ensemble $\mathcal{C}$ des classes d'équivalence peut être construit facilement.

Dans ce qui suit, nous allons présenter une méthode efficace permettant de caractériser les classes d'équivalence, puis nous allons montrer comment construire efficacement l'ensemble $\mathcal{C}$. La proposition suivante permet de caractériser nos classes d'équivalence :

Proposition 3.5.

1. $C_0 = \{\pi_X\sigma_y(r) \in \mathcal{Q} \mid y \notin \pi_Y(r)\}$.
2. $C_1 = \{\pi_X\sigma_y(r) \in \mathcal{Q} \mid y \in \pi_Y(r), Y \rightarrow X \in \mathcal{F}^+\}$.
3. *Pour toute requête $q = \pi_X\sigma_y(r)$ de $\mathcal{Q} \setminus (C_0 \cup C_1)$, $[q]$ est l'ensemble de toutes les requêtes $q_1 = \pi_{X_1}\sigma_{y_1}(r_1)$ de $\mathcal{Q}$ telles que :*

- *Si r est une table de dimension alors $r_1 = r$, sinon r_1 est un* kfk-join *défini sur le schéma R_1 tel que $X_1 \subseteq R_1$ et $Y_1 \subseteq R_1$*
- *$(X_1Y_1)^+ = (XY)^+$ et $Y_1^+ = Y^+$*
- *$yy_1 \in \pi_{YY_1}(r \bowtie r_1)$.*

Démonstration. 1, 2. Si $[q] = C_0$ (respectivement $[q] = C_1$), le résultat vient directement de la Définition 3.5.(1) (respectivement de la Définition 3.5.(2)).
3. Si la classe $[q]$ est différente de C_0 et C_1, soit une requête $q_1 = \pi_{X_1}\sigma_{y_1}(r_1)$ telle que $q \equiv q_1$.
Comme $q \preceq q_1$ et $q_1 \preceq q$, $r \lhd r_1$ et $r_1 \lhd r$ sont vérifiés.
Ainsi, si r est une table de dimension alors $r = r_1$, sinon, r_1 est un *kfk-join*.

De plus, les dépendances fonctionnelles $XY_1 \rightarrow X_1$, $Y_1 \rightarrow Y$, $X_1Y \rightarrow X$ et $Y \rightarrow Y_1$ sont dans $\mathcal{F}^+$, ce qui implique que $(X_1Y_1)^+ = (XY)^+$ et $Y_1^+ = Y^+$.
Nous avons également $yy_1 = \pi_{YY_1}(r \bowtie r_1)$, d'où q_1 satisfait la Proposition 3.5.(3).
Inversement, si q_1 satisfait l'item 3 de la proposition, alors $X_1Y_1 \rightarrow XY$, $XY \rightarrow X_1Y_1$, $Y \rightarrow Y_1$ et $Y_1 \rightarrow Y$ sont dans $\mathcal{F}^+$.
Ainsi, en utilisant les axiomes d'Armstrong, nous déduisons que les dépendance $XY_1 \rightarrow X_1$ et $X_1Y \rightarrow X$ sont dans $\mathcal{F}^+$.
De plus, comme $yy_1 \in \pi_{YY_1}(r \bowtie r_1)$, en utilisant la Définition 3.5.(3), nous avons $q \preceq q_1$ et $q_1 \preceq q$, ce qui implique que $q \equiv q_1$. Ce qu'il fallait démontrer. □

Il faut noter, grâce à la Proposition 3.5, que les classes C_0 et C_1 deviennent faciles à caractériser. Il est à remarquer également que même si C_0 et C_1 contiennent une infinité de requêtes, ces deux classes sont toujours non fréquentes.
Ainsi, dans le reste de ce chapitre, nous porterons notre attention sur les classes de requêtes différentes de C_0 et C_1, qui sera noté $\mathcal{C}^*$.
Le Corrollaire 3.2 montre que chaque classe d'équivalence $[q] \in \mathcal{C}^*$ admet un représentant unique q_0.

Corollaire 3.2. *Soit $[q]$ une classe de requêtes de $\mathcal{C}^*$ telle que $q = \pi_X\sigma_y(r)$.*
Alors, la requête $q_0 = \pi_{X_0}\sigma_{y_0}(r_0)$ définie de la manière suivante est dans $[q]$:
- *Si r est une table de dimension, alors $r_0 = r$ sinon $r_0 = \delta_1 \bowtie \ldots \bowtie \delta_N \bowtie \varphi$.*
- *$X_0 = (XY)^+$ et $Y_0 = Y^+$, et*
- *y_0 est un tuple sur Y_0 appartenant à $\pi_{Y_0}(r_0)$ et tel que $y_0.Y = y$.*

De plus, q_0 est unique.

Démonstration. Ce corollaire est une conséquence directe de la Proposition 3.5.
En effet, q_0 vérifie :
- si r est une table de dimension :
  - $r_0 = r$,
  - $X_0Y_0 = ((XY)^+Y^+)$ car comme $(Y \subseteq XY)$ nous avons $X_0Y_0 = (XY)^+$ et $Y_0 = Y^+$,
  - Comme $y_0.Y = y$, nous avons $y \in \pi_Y(r_0) = \pi_Y(r \bowtie r_0)$ car $r = r_0$, d'où $yy_0 \in \pi_{YY_0}(r_0)$.
- si r n'est pas une table de dimension :

– il est facile de vérifier que $X_0 \subseteq U$ et $Y_0 \subseteq U$ et que $r_0 = \delta_1 \bowtie \ldots \bowtie \delta_N \bowtie \varphi$ est un *kfk-join*,
– Puisque $(Y \subseteq XY)$, $X_0Y_0 = ((XY)^+Y^+)$ ce qui implique que $X_0Y_0 = (XY)^+$ et $Y_0 = Y^+$,
– comme $y_0.Y = y$, nous avons $y \in \pi_Y(r_0) = \pi_Y(r \bowtie r_0)$ car $r \bowtie r_0 = r_0$, d'où $yy_0 \in \pi_{YY_0}(r_0) = \pi_{YY_0}(\delta_1 \bowtie \ldots \bowtie \delta_N \bowtie \varphi)$.

Par suite q_0 vérifie la Proposition 3.5.(3), et donc, nous en déduisons que q_0 est un représentant de la classe $[q]$. Montrons maintenant son unicité. Soit un autre représentant $q_1 = \pi_{X_1}\sigma_(Y_1)$ différent de q_0 et vérifiant les conditions du corollaire, nous avons alors $X_1 = (XY)^+$, $Y_1 = Y^+$ ce qui implique que $X_1 = X_0$ et $Y_1 = Y_0$, de même $y_1.Y_1 = y$ comme $Y_1 = Y_0$ alors $y_1.Y_0 = y$, nous en déduisons alors que $y_1.Y = y_0.Y$ ce qui implique que $y_1 = y_0$, ainsi $q_0 = q_1$ ce qui est absurde. D'où q_0 est unique. □

Notons par J la jointure $\delta_1 \bowtie \ldots \bowtie \delta_N \bowtie \varphi$, le Corollaire 3.2 montre que seulement deux types de classes requêtes sont à considérer : les classes de requêtes sur les tables de dimension et les classes de requêtes sur la jointure J.
En conséquence, le calcul peut être effectué en considérant tout d'abord les tables de dimension, puis la table de jointure J.
Nous allons donner dans la suite plus de détails pour le calcul des classes fréquentes, en attendant voici quelques exemples pour illustrer la Proposition 3.5 et le Corollaire 3.2.

Exemple 3.5. *Reprenons notre base de données référence de la Figure 3.1, nous avons* $J = (Cust \bowtie Prod \bowtie Sales)$.
Considérons, les deux requêtes $q_3 = \pi_{Cid\,Cname}\sigma_\top\,(Cust \bowtie Sales)$ *et* $q'_3 = \pi_{Cid\,Cname\,Caddr}\sigma_\top(J)$ *de l'Exemple 3.4, qui sont comme nous l'avons vu, équivalentes.*
Comme $(Cid\,Cname)^+ = (Cid\,Cname\,Caddr)$ *et* $\emptyset^+ = \emptyset$, *la classe* $[q_3]$ *sera représentée par* q'_3, *et, en utilisant la Proposition 3.5.3,* $[q_3]$ *est l'ensemble de toutes les requêtes* $\pi_X\sigma_\top(r)$ *telles que* $Cid \subseteq X \subseteq (Cid\,Cname\,Caddr)$ *et soit* $r = J$ *ou* $r = (Cust \bowtie Sales)$.
Considérons la requête $q = \pi_{Cname\,Ptype}\sigma_{\mathtt{p}_2}(J)$, *nous avons :*

– $(Cname\,Pid\,Ptype)^+ = (Cname\,Pid\,Ptype)$,
– $Pid^+ = (Pid\,Ptype)$, *et*
– $\mathtt{p}_2\,\mathtt{beer} \in \pi_{Pid\,Ptype}(J)$.

Ainsi, la classe $[q]$ *est représentée par* $\pi_{Cname\,Pid\,Ptype}\sigma_{\mathtt{p}_2\,\mathtt{beer}}(J)$, *et cette classe est l'ensemble de toutes les requêtes* $\pi_X\sigma_y(r)$ *telles que* $Cname\,Ptype \subseteq X \subseteq (Cname\,Pid\,Ptype)$, *et*

– *soit* $y = \mathtt{p}_2$ *avec* $r = (Cust \bowtie Sales)$ *ou* $r = J$,
– *soit* $y = \mathtt{p}_2\,\mathtt{beer}$ *avec* $r = J$.

3.3.4 Calcul de $\mathcal{C}^*$

Dans la suite de ce chapitre, la relation r représentera une table de dimension δ_i ou la jointure $J = \delta_1 \bowtie \delta_2 ... \bowtie \varphi$. Nous rappelons que $cl(\mathcal{F})$ représente l'ensemble de tous les schémas X tels que $X^+ = X$, où X^+ représente la fermeture de X, également, dans le cas d'un schéma étoile ayant N dimensions, si K est l'union des clés des tables de dimension (*i.e.,* $K = K_1 \ldots K_N$), X est dans $cl(\mathcal{F})$ si et seulement si $X = R$, ou $K \not\subseteq X$ et pour

toute clé $K_i \in X$, $D_i \subseteq X$, où R représente le schéma d'une relation r de la base de données.
Soit un schéma X de $cl(\mathcal{F})$ tels que $X \subseteq R$, nous noterons dans la suite de ce chapitre que $X_R^{\downarrow}$ (respectivement $X_R^{\uparrow}$) représente l'ensemble des schémas maximaux (respectivement minimaux) X' de $cl(\mathcal{F})$ tels que $X' \subset X$ (respectivement $X \subset X' \subseteq R$).
Dans le cas d'un schéma étoile, nous montrons que :

- $R^{\downarrow} = \{R \setminus A : A \in K\}$, et $\emptyset^{\uparrow} = \{A \in R : A \notin K\}$,
- si $X \subset R$ et $X \neq \emptyset$, $X_R^{\downarrow}$ est l'ensemble de tous les schémas $X \setminus A$ où A est un attribut de X tel que :
  - soit $A \in K$,
  - soit ($K_i \notin X$ et $A \in D_i M$, pour un $i \in \{1, \ldots, N\}$).
- Si $X \subset R$ et $X \neq \emptyset$, $X_R^{\uparrow}$ est l'ensemble de tous les schemas XA où A est un attribut de $(R \setminus X)$ tel que :
  - $A \notin K$ ou,
  - $\exists i \in \{1, \ldots, N\}$ tel que $A = K_i$, $(D_i \setminus K_i) \subseteq X$, et si $K \subseteq XK_i$ alors $M \subseteq X$.

Remarque 3.2. *$R^{\uparrow}$ et $\emptyset^{\downarrow}$ ne sont pas définis.*

Exemple 3.6. *Pour illustrer nos propos, dans le contexte de notre exemple de référence (Figure 3.1), soit $U = Cid\,Cname\,Pid\,Ptype\,Qty$ nous avons les égalités suivantes :*

- $(Cname\,Pid\,Ptype)_U^{\downarrow} = \{(Pid\,Ptype)$,
- $(Ptype)_{Pid\,Ptype}^{\uparrow} = \{(Pid\,Ptype)\}$,
- $(Pid\,Ptype)_{Pid\,Ptype}^{\downarrow} = \{(Ptype)\}$,
- $\emptyset_{Pid\,Ptype}^{\uparrow} = \{(Ptype)\}$.

Dans la suite de cette contribution, pour simplifier la notation, chaque classe de $[q]$ de $\mathcal{C}^*$ sera notée tout simplement q où q est le représentant de la classe tel que défini dans le Corollaire 3.2.
Soit une classe $q = \pi_X \sigma_y(r)$ de $\mathcal{C}^*$, l'ensemble de tous les successeurs de q, noté $succ(q)$, est l'ensemble des classes $q' = \pi_{X'} \sigma_{y'}(r')$ de $\mathcal{C}^*$ telles que $r = r'$, $q \preceq q'$, et pour toute classe q'' telle que $q \preceq q'' \preceq q'$, soit $q'' = q$ ou $q'' = q'$.

La proposition suivante permet de caractériser les successeurs d'un classe de requête q dans le cas d'une base de données définie sur un schéma étoile.

Proposition 3.6. *Soient une relation r définie sur le schéma R, et $[q]$ une classe de $\mathcal{C}^*$, ayant pour représentant la requête $q = \pi_X \sigma_y(r)$. Alors, $succ([q])$ est l'ensemble de toutes les classes $[q']$ ayant pour représentant $q' = \pi_{X'} \sigma_{y'}(r)$ définies de la manière suivante :*

1. *$succ(q) = \{C_1\}$ si et seulement si $Y_R^{\uparrow} = \{X\}$.*
2. *Si $A \in (X \setminus Y)$ est tel que $Y \subset (X \setminus A)$ et $(X \setminus A) \in X_R^{\downarrow}$, alors $X' = (X \setminus A)$ et $y' = y$.*
3. *Si A est tel que $A \in (X \setminus Y)$, $YA \in Y_R^{\uparrow} \setminus \{X\}$ et $(X \setminus A) \notin (cl(\mathcal{F}) \setminus \{\emptyset\})$, alors $X' = X$, $Y' = YA$ et $y' = ya$ où $ya \in \pi_{YA}(r)$.*
4. *Si A est tel que $A \in (R \setminus X)$, $YA \in Y_R^{\uparrow}$ et y' est tout tuple sur Y' tel que $y' = ya$ et $ya \in \pi_{YA}(r)$, alors*

(a) si $A \in K$, $(K \setminus A) \subseteq X$ *et* $(M \cap (X \setminus Y)) = \emptyset$, *alors* $X' = R$ *et* $Y' = YA$,

(b) si ($A \in K$ *et* $(K \setminus A) \not\subseteq X$ *et* $XA \in X_R^{\uparrow}$) *ou* ($A \notin K$), *alors* $X' = XA$ *et* $Y' = YA$.

Démonstration. Les items de cette proposition correspondent respectivement à ceux de la Proposition A.1.
Nous allons démontrer chaque item dans la suite.

1. Comme chaque $Y' \in Y^{\uparrow}$ est de la forme YA avec $A \notin Y$, et comme $Y \subset X$, $X \subseteq YA$ est équivalent à $X = YA$. Le résultat vient alors de l'item 1 de la Proposition A.1.

2. Il est facile de voir que l'item 2 de la Proposition A.1 correspond à l'item 2 de la Proposition A.1.

3. Il est facile de voir que l'items 3(a) et 3(b) de la Proposition A.1 correspond à $A \in (X \setminus Y)$ et $YA \in Y^{\uparrow} \setminus \{X\}$, respectivement.
Ainsi, nous allons démontrer que l'item 3(c) de la Proposition A.1 est équivalent à $(X \setminus A) \notin cl(FD) \setminus \{\emptyset\}$.
− Soit $A \in (X \setminus Y)$ tel que $(X \setminus A) \in cl(FD) \setminus \{\emptyset\}$. Alors $Y \subseteq (X \setminus A) \subset X \subseteq ((X \setminus A)A)^+$, ce qui montre, avec $X_0 = (X \setminus A)$, item 3(c) de la Proposition A.1 n'est pas vérifié.
− Inversement, si l'item 3(c) de la Proposition A.1 n'est pas satisfait, alors soit X_0 appartenant à $cl(FD) \setminus \{\emptyset\}$ tel que $Y \subseteq X_0 \subset X \subseteq (X_0A)^+$.
Dans ce cas, $A \notin X_0$ car, autrement, nous aurions $X_0 \subset X \subseteq X_0$.
Si $(X_0A)^+ = X_0A$, alors nous avons $X_0 = (X \setminus A)$, ce qui montre que $(X \setminus A) \in cl(FD) \setminus \{\emptyset\}$.
Si $(X_0A)^+ \neq X_0A$, alors comme $X_0 \in cl(FD) \setminus \{\emptyset\}$, nous avons A appartient à K. Soit $A = K_i$ ($i \in \{1, \ldots, N\}$), comme $K_i \in X$ et $X \in cl(FD)$, $D_i \subseteq X$.
Ainsi, $(X \setminus K_i) \in cl(FD) \setminus \{\emptyset\}$, ce qui montre que $(X \setminus A) \in cl(FD) \setminus \{\emptyset\}$.

4.Il est facile de voir que l'items 4(a) et 4(b) de la proposition Proposition A.1 correspond à $A \notin X$ et $YA \in Y^{\uparrow}$, respectivement.
Ainsi, nous allons démontrer que l'item 4(c) de la Proposition A.1, est équivalent à :
(a) $((M \cap (X \setminus Y)) = \emptyset$ et $(K \setminus A) \subseteq X$ et $A \in K)$ ou
(b) ($A \in K$ et $(K \setminus A) \not\subseteq X$ et $XA \in X^{\uparrow}$) ou ($A \notin K$).

Supposons tout d'abord que (a) ni (b) soit vérifié. Ce qui implique que $A \in K$, soit $i \in \{1, \ldots, N\}$ tel que $A = K_i$. Ainsi, la formule (1) suivante est vérifiée :
$((M \cap (X \setminus Y)) \neq \emptyset$ ou $(K \setminus K_i) \not\subseteq X)$ et
$((K \setminus K_i) \subseteq X$ ou $XK_i \notin X^{\uparrow})$.
Nous notons tout d'abord que $((K \setminus K_i) \not\subseteq X$ et $XK_i \notin X^{\uparrow})$ n'est pas possible. Ainsi, comme $(K \setminus K_i) \not\subseteq X$, si $XK_i \notin X^{\uparrow}$ alors $(D_i \setminus K_i) \not\subseteq X$. Ainsi, $(D_i \setminus K_i) \not\subseteq Y$, ce qui n'est pas possible d'où $YK_i \in Y^{\uparrow}$. Ainsi, la formule (1) se réduit à :
$((M \cap (X \setminus Y)) \neq \emptyset)$ et $((K \setminus K_i) \subseteq X$ ou $XK_i \notin X^{\uparrow})$.
− Si $((M \cap (X \setminus Y)) \neq \emptyset$ et $(K \setminus K_i) \subseteq X)$ sont vérifié, pour $X_0 = (X \setminus (M \cap (X \setminus Y)))$, nous avons $X_0 \in cl(FD)$, $Y \subseteq X_0 \subset X$ et $(K \setminus K_i) \subseteq X_0$. Ainsi, $(XK_i)^+ = (X_0K_i)^+ = U$, ce qui montre que l'item 4(c) de la Proposition A.1 n'est pas vérifié.
− Si $((M \cap (X \setminus Y)) \neq \emptyset$ et $XK_i \notin X^{\uparrow})$ sont vérifiés, alors nous avons $(K \setminus K_i) \subseteq X$ car $((K \setminus K_i) \not\subseteq X$ et $XK_i \notin X^{\uparrow})$ n'est pas possible. Ce qui montre que ce cas est similaire au cas précédent.

Inversement, si l'item 4(c) de la Proposition A.1 n'est pas satisfaite, alors X_0 appartient à $cl(FD)$ tel que $Y \subseteq X_0 \subset X$ et $(XA)^+ = (X_0A)^+$. Si $A \notin K$ alors, comme $A \notin X$, nous avons $A \notin X_0$, et ainsi, $(XA)^+ = XA$ et $(X_0A)^+ = X_0A$. Nous en déduisons que, $(XA)^+ \subset (X_0A)^+$, ce qui est impossible.

Soit $A = K_i$ pour i appartenant à $\{1, \ldots, N\}$ comme ci dessus.
Alors, comme $YK_i \in Y^{\uparrow}$, nous avons $(D_i \setminus K_i) \subseteq Y$ et ainsi, $(D_i \setminus K_i) \subseteq X$ et $(D_i \setminus K_i) \subseteq X_0$.
(a) Si $(K \setminus K_i) \subseteq X$, alors $(XK_i)^+ = U$, et ainsi, $(XK_i)^+ = (X_0K_i)^+ = XK_iM$. De plus, comme X et X_0 sont dans $cl(FD)$, pour tout $j \neq i$, $D_j \subseteq X$ et $D_j \subseteq X_0$.
Ainsi, $(X \setminus X_0) \subseteq M$, et comme $Y \subseteq X_0 \subset X$, nous avons $(X \setminus X_0) = ((X \setminus Y) \setminus X_0)$.
Ainsi, $M \cap (X \setminus Y) \neq \emptyset$.
(b) Si $(K \setminus K_i) \not\subseteq X$, alors nous avons également $(K \setminus K_i) \not\subseteq X_0$.
Ainsi, $(XK_i)^+ = XK_i$ et $(X_0K_i)^+ = X_0K_i$, ce qui est en contradiction avec $(XK_i)^+ = (X_0K_i)^+$.

Ainsi, la Proposition A.1(4) n'est pas satisfaite dans ce cas, ce qui complète la preuve.
Nous rappelons que si q est de la forme $\sigma_X\sigma_y(J)$, alors $R = U$ et les classes considérées sont celle définies précédemment.
Le résultat provient de la Proposition A.1 qui permet de caractériser $succ(q)$ dans le cas d'une seule table vérifiant un ensemble de dépendances fonctionnelles.Il faut noter cependant que dans ce cas l'item 4(a) ne s'applique pas.

Supposons maintenant que q est de la forme $\sigma_X\sigma_y(\delta_i)$, où δ_i est une table de dimension. Dans ce cas, $R = D_i$ et les seules dépendances fonctionnelles qui nous intéressent sont $K_i \rightarrow D_i$ et leurs conséquences en utilisant les axiomes de Armstrong.

□

Exemple 3.7. *Dans le contexte de notre base de données de référence (Figure 3.1), illustrons la Proposition 3.6 en calculant succ(q), avec* $q = \pi_{Cname\,Caddr}\sigma_{\texttt{Paris}}(Cust)$. *Dans ce cas,* $r = Cust$ *ainsi,* $R = (Cid\,Cname\;Caddr)$ *et* $Caddr_R^{\uparrow} = \{(Cname\,Caddr)\}$. *Ainsi, en utilisant la Proposition 3.6.(1), nous obtenons* $succ(q) = \{C_1\}$.
Considérons maintenant la classe de requêtes $\pi_{Cid\,Cname\,Caddr}\sigma_{\top}(Cust)$, *nous avons* $r = Cust$, $R = (Cid\,Cname\,Caddr)$, *mais, la Proposition 3.6.(1) ne s'applique pas comme*
$\emptyset_R^{\uparrow} = \{Cname, Caddr\}$
$\neq R$.

Comme $R_R^{\downarrow} = \{(Cname\,Caddr)\}$, *la Proposition 3.6.(2) s'applique pour* $A = Cid$.
Ainsi, $\pi_{Cname\,Caddr}\sigma_{\top}(Cust) \in succ(q)$.

De même la Proposition 3.6.(3) s'applique pour $A = Cname$ *et* $A = Caddr$ *car dans ce cas,* $A \in \emptyset_R^{\uparrow}$, $A \neq X$ *et* $((Cid\,Cname\;Caddr) \setminus A) \notin (cl(\mathcal{F}) \setminus \{\emptyset\})$. *Ainsi,* $succ(q) = \{\pi_R\sigma_{\texttt{John}}(Cust), \pi_R\sigma_{\texttt{Mary}}(Cust), \pi_R\sigma_{\texttt{Jane}}(Cust), \pi_R\sigma_{\texttt{Anne}}(Cust), \pi_R\sigma_{\texttt{Paris}}(Cust), \pi_R\sigma_{\texttt{Tours}}(Cust)\}$.

Finalement, la Proposition 3.6.(4) ne s'applique pas, car $(Cid\,Cname\,Caddr) = R$.
Ainsi, nous avons : $succ(q) =$
$\{\pi_X\sigma_{\top}(Cust), \pi_R\sigma_{\texttt{John}}(Cust), \pi_R\sigma_{\texttt{Mary}}(Cust), \pi_R\sigma_{\texttt{Jane}}(Cust), \pi_R\sigma_{\texttt{Anne}}(Cust),$
$\pi_R\sigma_{\texttt{Paris}}(Cust), \pi_R\sigma_{\texttt{Tours}}(Cust)\}$, *avec* $X = R \setminus Cid = (Cname\,Caddr)$.

Dans la section suivante, nous allons présenter notre algorithme de recherche de requêtes fréquentes.

3.4 Algorithmes

Dans le but de ne pas générer toutes les classes candidates, nous définissons la notion de *classe générique* de la manière suivante :

Définition 3.6. *Soit une classe $q = \pi_X \sigma_y(r)$ in $\mathcal{C}^*$, la* classe générique *associée à q, notée $\langle X, Y, r\rangle$, est l'ensemble de toutes les classes $\pi_X \sigma_{y'}(r)$ de $\mathcal{C}^*$ tel que y' est un tuple de $\pi_Y(r)$,* i.e., $\langle X, Y, r\rangle = \{\pi_X \sigma_{y'}(r) \in \mathcal{C}^* \mid y' \in \pi_Y(r)\}$.

La définition suivante permet de caractériser les successeurs d'une classe générique $\langle X, Y, r\rangle$.

Définition 3.7. *Soit une classe générique $\langle X, Y, r\rangle$, où la table r défini sur le schéma R est une table de dimension ou la table de jointure, X et Y deux schémas de R . Alors, $succ(\langle X, Y, r\rangle)$ est l'ensemble de toutes les classes génériques $\langle X', Y', r\rangle$ définies de la manière suivante :*

1. *$succ(\langle X, Y, r\rangle) = \{C_1\}$ si et seulement si $Y_R^{\uparrow} = \{X\}$.*
2. *Si $A \in (X \setminus Y)$ est tel que $Y \subset (X \setminus A)$ et $(X \setminus A) \in X_R^{\downarrow}$, alors $X' = (X \setminus A)$.*
3. *Si A est tel que $A \in (X \setminus Y)$, $YA \in Y_R^{\uparrow} \setminus \{X\}$ et $(X \setminus A) \notin (cl(\mathcal{F}) \setminus \{\emptyset\})$, alors $X' = X$, $Y' = YA$.*
4. *Si A est tel que $A \in (R \setminus X)$, $YA \in Y_R^{\uparrow}$ et y' est tout tuple sur Y' alors :*
   (a) *si $A \in K$, $(K \setminus A) \subseteq X$ et $(M \cap (X \setminus Y)) = \emptyset$, alors $X' = R$ et $Y' = YA$,*
   (b) *si ($A \in K$ et $(K \setminus A) \not\subseteq X$ et $XA \in X_R^{\uparrow}$) ou ($A \notin K$), alors $X' = XA$ et $Y' = YA$.*

Proposition 3.7. *Soit une classe de requête $q = \pi_X \sigma_y(r)$: $\{succ(q) : q \in \langle X, Y, r\rangle\} = succ(\langle X, Y, r\rangle)$.*

Démonstration. Voir Annexe A.2.

□

Dans la suite, grâce à la Proposition 3.7, nous considérerons, lors de la phase de génération, les classes génériques de requêtes à la place des classes de requêtes pour des raisons d'optimisation. En effet, comme nous pouvons le remarquer les classes génériques de requêtes ne dépendent pas de la taille de la base de données considérée contrairement aux classes de requêtes. Une classe générique $\langle X, Y, r\rangle$ sera dite fréquente si elle contient au moins une classe de requêtes fréquente. Ainsi, si elle ne contient pas une classe de requête fréquente, inutile de générer ses successeurs qui sont non fréquentes à cause de la propriété d'anti-monotonie du support.
L'ensemble des successeurs de la classe générique $\langle X, Y, r\rangle$, notée $succ(\langle X, Y, r\rangle)$, est calculé en utilisant la Proposition 3.7 comme l'illustre l'Exemple 3.8.

Exemple 3.8. *La classe générique associée à $\pi_X \sigma_\top(Cust)$ où $X = (Cid\, Cname\, Caddr)$, noté $\langle X, \emptyset, Cust\rangle$, l'ensemble $succ(\langle CidCnameCaddr, \emptyset, Cust\rangle) = \{\langle CnameCaddr, \emptyset, Cust\rangle$, $\langle CidCnameCaddr$,*
$Cname$, $Cust\rangle$, $\langle CidCnameCaddr, Caddr, Cust\rangle\}$.

3.4.1 Algorithme Principal : FQF

Les classes fréquentes de $\mathcal{C}^*$ sont calculées grâce à un algorithme par niveau, que nous avons baptisé *Frequent Query Finder* (FQF), dont les étapes principales sont représentées par l'Algorithme 9.
Cet algorithme retourne l'ensemble des classes fréquentes $Freq$ de $\mathcal{C}^*$. L'algorithme FQF utilise principalement l'algorithme mine (ligne 3 et ligne 7). L'algorithme mine permet de rechercher les requêtes fréquentes d'une table r donnée (r étant une table de dimension ou la table de jointure J). Nous présenterons plus en détail cet algorithme dans la section suivante. Comme le montre l'Algorithme 9, la recherche est effectuée tout d'abord au niveau des tables de dimension (ligne 3), puis au niveau de la table de jointure J (ligne 7).

Algorithm 9 FQF

Entrées: Une base de données Δ ayant un schéma étoile de N-dimensions et un seuil de support *min-sup*.
Sortie: L'ensemble $Freq$ des classes de requêtes fréquentes.

1: $Freq = \emptyset$
2: **Pour Tout** $i = 1, \ldots, N$ **do**
3: mine(δ_i, $Freq(\delta_i)$)
4: $Freq = Freq \cup Freq(\delta_i)$
5: **Fin Pour Tout**
6: compute $J = \delta_1 \bowtie \ldots \bowtie \delta_N \bowtie \varphi$
7: mine(J, $Freq(J)$)
8: $Freq = Freq \cup Freq(J)$
9: **Retourner** $Freq$

3.4.2 Algorithme mine

L'Algorithme mine, représenté par la Figure 10, permet de calculer les requêtes fréquentes d'une table r donnée ayant pour schéma R, r étant une table de dimension ou la table de Jointure J d'une base de données Δ ayant un schéma étoile. Cet algorithme utilise principalement les algorithmes :

- generate : cet algorithme permet de générer les classes génériques candidates.
- prune : cet algorithme permet d'élaguer l'ensemble des classes génériques candidates.
- scan : cet algorithme permet d'évaluer le support des classes de requêtes candidates.

Nous présenterons en détail chacun de ces algorithmes dans les section suivantes.
L'algorithme FQF suit un parcours par niveau comme Apriori ([8]). Plus précisément, pour chaque table de la base de données et de la table de jointure r ayant pour schéma R, FQF commence par la classe générique de requêtes la plus générale, c'est à dire $\langle R, \emptyset, r \rangle$, puis les étapes suivantes seront effectuées tant qu'il existe des classes fréquentes.

1. Générer et élaguer l'ensemble C des classes génériques candidates, en utilisant la liste courante L des classes génériques fréquentes (c'est à dire contenant au moins une requête fréquente) ;

2. Evaluer le support des classes de C non élaguées ;
3. Supprimer les classes dont les supports sont inférieurs au seuil minimal ;
4. Mettre dans L les classes génériques qui contiennent au moins une classe fréquente.

Cependant, les étapes ci-dessous demandent plus d'attention que l'algorithme Apriori [8] car :

- (i) Nous travaillons avec des classes d'équivalence, à la place des itemsets,
- (ii) l'ordre sur $\mathcal{C}^*$ est plus difficile à tester que l'inclusion ensembliste, et
- (iii) le calcul du support requiert un parcours efficace de la base.

En conséquence, les difficultés sont tout d'abord, la génération et l'élagage des classes candidates, et la seconde, le calcul efficace des supports de $\mathcal{C}^*$.
Le premier point est traité par la Proposition 3.7, la seconde difficulté sera quant à elle traitée dans la Section 3.4.5.
A chaque étape, l'Algorithme `mine` calcule les classes de requêtes fréquentes du niveau courant ($LFreq(r)$). A la fin de chaque étape, $LFreq(r)$ est ajouté à l'ensemble des classes fréquentes de la table r ($Freq(r)$). Enfin, l'algorithme renvoie les classes de requêtes fréquentes $Freq(r)$ de la table r.

Algorithm 10 `mine`

Entrées: Une table r (une table de dimension δ_i ou une table de jointure J) définie sur le schéma R.

Sortie: L'ensemble $Freq(r)$ est l'ensemble des classes fréquentes de $\mathcal{C}^*$ de la forme $\pi_X\sigma_y(r)$.

Si $|r| <$ *min-sup* **Alors**
  //Aucune requête n'est fréquente
  $Freq(r) = \emptyset$
Sinon
  //Le calcul commence avec la classe générique $\langle R, \emptyset, r\rangle$
  $L = \{\langle R, \emptyset, r\rangle\}$
  $Freq(r) = \{\pi_R\sigma_\top(r)\}$
  Tant Que $L \neq \emptyset$ **do**
    //L est l'ensemble des classes génériques fréquentes du niveau précédent.
    $C =$ `generate`(L, r)
    $C =$ `prune`(C, L, r)
    `scan`$(C, \text{AUX}(r), L, L_{Freq}(r))$
    //L contient l'ensemble des classes génériques fréquentes du niveau courant.
    //$L_{Freq}(r)$ correspond à l'ensemble des requêtes fréquentes.
    $Freq(r) = Freq(r) \cup L_{Freq}(r)$
  Fin Tant Que
  Retourner $Freq(r)$
Fin Si

La génération des classes de requêtes est effectuée grâce à l'algorithme `generate` décrit ci dessous.

3.4.3 Algorithme `generate`

L'algorithme de génération (`generate`) des classes candidates utilisé dans l'algorithme `mine` est représenté par l'Algorithme 11. Cette algorithme permet de générer, grâce à la Définition 3.7(2) et la Proposition 3.7, les classes candidates du niveau $i+1$ noté C, à partir des classes fréquentes du niveau i L.

Algorithm 11 `generate`

Entrées: Une table r, l'ensemble L des classes génériques fréquentes du niveau l.
Sortie: L'ensemble C des classes candidates du niveau $l+1$ obtenu grâce à l'ensemble L.

1: $C = \emptyset$
2: **Pour Tout** $\langle X, Y, r\rangle \in L$ **do**
3: **Si** $Y^{\uparrow} = \{X\}$ **Alors**
4: //Définition 3.7(1) et Proposition 3.7
5: $C = C_1$
6: **Sinon**
7: **Pour Tout** $A \in (X \setminus Y)$ **do**
8: **Si** $(X \setminus A) \in (X^{\downarrow} \setminus \{Y\})$ **Alors**
9: //Définition 3.7(2) et Proposition 3.7
10: $C = C \cup \{\langle (X \setminus A), Y, r\rangle\}$
11: **Fin Si**
12: **Si** $(X \setminus A) \notin cl(F) \setminus \{\emptyset\}$ `et` $YA \in Y^{\uparrow}$ **Alors**
13: //Définition 3.7(3) et Proposition 3.7
14: $C = C \cup \{\langle X, YA, r\rangle\}$
15: **Fin Si**
16: **Fin Pour Tout**
17: **Pour Tout** $A \in (U \setminus X)$ **do**
18: **Si** $YA \in Y^{\uparrow}$ `et` $A \in K$ `et` $(K \setminus A) \subseteq X$ `et` $(M \cap (X \setminus Y)) = \emptyset$ **Alors**
19: //Définition 3.7(4.a) et Proposition 3.7.
20: $C = C \cup \{\langle U, YA, r\rangle\}$
21: **Fin Si**
22: **Si** $YA \in Y^{\uparrow}$ `et ((`$A \in K$ `et` $(K \setminus A) \not\subseteq X$`et` $XA \in X^{\uparrow}$`) ou (`$A \notin K$`))` **Alors**
23: //Définition 3.7(4.b) et Proposition 3.7.
24: $C = C \cup \{\langle XA, YA, r\rangle\}$
25: **Fin Si**
26: **Fin Pour Tout**
27: **Fin Si**
28: **Fin Pour Tout**
29: **Retourner** C

3.4.4 Algorithme `prune`

La classe des requêtes candidates C est élaguée en utilisant la Proposition 3.4. En effet, grâce à la propriété d'anti-monotonie, si une classe candidate $\langle X, Y, r\rangle$ est telle

que $\langle X, Y, r\rangle \in succ(\langle X', Y', r\rangle)$ et $\langle X', Y', r\rangle$ non fréquente, alors $\langle X, Y, r\rangle$ ne peut être fréquente. L'algorithme prune est basé sur la Proposition 3.8.
Il est à noter que l'élagage est effectué à deux niveaux : d'abord l'élagage de l'ensemble des classes génériques candidates (ligne 11 de l'Algorithme `mine`) et l'élagage de l'ensemble des classes de requêtes candidates (ligne 19 de l'Algorithme `scan`).
Dans l'Algorithme `mine`, la classe générique $\langle X, Y, r\rangle$ est élaguée si au moins un de ses prédécesseurs (suivant $\preceq$) ne contient aucune classe fréquente, car dans ce cas aucune classe de $\langle X, Y, r\rangle$ ne peut être fréquente. Cette élagage est effectuée grâce à la fonction `prune`.
Cependant, si $\langle X, Y, r\rangle$ n'a pas été élaguée, il se pourrait qu'une classe particulière $\pi_X\sigma_y(r)$ de $\langle X, Y, r\rangle$ puisse être élaguée. De plus, il est important de noter qu'un élagage additionnel est possible.

En effet, les classes de requêtes qui sont de la forme $\pi_X\sigma_y(J)$, avec $X \subseteq D_i$, sont telles que $\pi_X\sigma_y(\delta_i) \leq \pi_X\sigma_y(J)$, avec $i \in \{1, ...n\}$ grâce aux dépendances d'inclusion $\mathcal{I}$. Cet élagage supplémentaire est effectué dans l'Algorithme `scan`, en utilisant l'Algorithme `pruneQuery` représenté par l'Algorithme 12.

Algorithm 12 `pruneQuery`

Entrées: A class $q = \pi_X\sigma_y(r)$.
Sortie: boolean.

1: **Si** ($\exists A \in Y$ et $a \in dom(A)$ tel que $q = \pi_X\sigma_{y'}(r) \notin L_{Freq}(r)$ et $y = y'a$) ou ($\exists i X \subseteq D_i, r = J$ et $\pi_X\sigma_y(\delta_i) \notin Freq(\delta_i)$) **Alors**
2: **Retourner** true ;
3: **Fin Si**
4: **Retourner** false ;

Toutefois, il est important de noter que notre élagage n'est pas complète, car on ne teste pas l'ensemble des prédécesseurs de $\pi_X\sigma_y(r)$ pour des raisons d'efficacité.

Proposition 3.8. *Soit une classe générique $\langle X, Y, r\rangle \in \mathcal{C}^*$, où la table r défini sur le schéma R est une table de dimension ou la table de jointure, X et Y deux schémas de R . Alors, l'ensemble des prédecesseurs de $\langle X, Y, r\rangle$ noté $pred(\langle X, Y, r\rangle)$ est l'ensemble de toutes les classes génériques $\langle X', Y', r\rangle$ définies de la manière suivante :*

1. *Si $(A \notin X)$ et $(XA) \in X_R^{\downarrow}$, alors $X' = (X \setminus A)$, $Y' = Y$.*
2. *Si $(A \in Y)$ et $(X \setminus A \notin (cl(\mathcal{F}) \setminus \{\emptyset\}))$ et $Y \setminus A \in Y_R^{\downarrow}$ alors $X' = X$, $Y' = Y \setminus A$.*
3. *Si $(A \in Y)$ et $((Y \setminus A) \in Y_R^{\downarrow}$, $A \in K$ `et` $X = U)$ alors $X' = \langle (R \setminus ((M \setminus Y)A))$ et $Y' = Y \setminus A$.*
4. *Si $(A \in Y)$ et $((Y \setminus A) \in Y_R^{\downarrow}$, si `((`$A \in K$ `et` $K \not\subseteq X$`) ou (`$A \notin K$`)) alors` $X' = X \setminus A$ et $Y' = Y \setminus A$.*

Démonstration. Cette proposition découle directement de la Définition 3.7. □

Algorithm 13 `prune`

Entrées: Une table r, l'ensemble L des classes génériques fréquentes du niveau l, l'ensemble C des classes génériques candidates obtenues à partir de L.

Sortie: L'ensemble C élagué.

1: // L'ensemble C est supposé différent de $\{\bot\}$.
2: **Pour Tout** $\langle X, Y, r\rangle \in C$ **do**
3: **Pour Tout** $A \notin X$ tel que $XA \in X^{\uparrow}$ **do**
4: **Si** $\langle XA, Y, r\rangle \notin L$ **Alors**
5: //Proposition 3.8(1)
6: $C = C \setminus \{\langle X, Y, r\rangle\}$
7: **Fin Si**
8: **Fin Pour Tout**
9: **Pour Tout** $A \in Y$ **do**
10: //Proposition 3.8(2)
11: **Si** $((X \setminus A) \notin cl(F) \setminus \{\emptyset\}$ `et` $(Y \setminus A) \in Y_R^{\downarrow}$ `et` $\langle X, (Y \setminus A), r\rangle \notin L)$ **Alors**
12: $C = C \setminus \{\langle X, Y, r\rangle\}$
13: **Fin Si**
14: //Proposition 3.8(3)
15: **Si** $((Y \setminus A) \in Y_R^{\downarrow}$ `et` $A \in K$ `et` $X = R$ `et` $\langle (U \setminus ((M \setminus Y)A)), (Y \setminus A), r\rangle \notin L)$ **Alors**
16: $C = C \setminus \{\langle X, Y, r\rangle\}$
17: **Fin Si**
18: //Proposition 3.8(4)
19: **Si** $((Y \setminus A) \in Y_R^{\downarrow}$ `et ((`$A \in K$ `et` $K \not\subseteq X$`) ou (`$A \notin K$`))` `et` $\langle (X \setminus A), (Y \setminus A), r\rangle \notin L)$ **Alors**
20: $C = C \setminus \{\langle X, Y, r\rangle\}$
21: **Fin Si**
22: **Fin Pour Tout**
23: **Fin Pour Tout**
24: **Retourner** C

3.4.5 Algorithme `scan`

Durant le scan de la table r, la principale difficulté est que chaque tuple de la réponse doit être compté juste *une fois*, mais, à cause de la projection, plusieurs tuples identiques peuvent apparaitre durant le parcours de la table r. Il faut alors un mécanisme qui permet de savoir si un tuple donné a déjà été rencontré ou pas.
Pour surmonter cette difficulté, une technique d'indexation serait nécessaire.
Mais malheureusement, une technique d'indexation qui prenne en compte toutes les valeurs possibles sur l'ensemble des attributs est irréaliste.
Nous proposons alors la solution suivante : avant de scanner la table r, nous construisons une table auxilliaire, notée $AUX(r)$, définie de la manière suivante. En supposant que r contient n tuples $t_1, \ldots, t_n$, le premier tuple (AUX(r)[1]) de AUX(r) est l'ensemble vide, et pour $i = 2, \ldots, n$, le i-eme élément de AUX(r), noté AUX(r)[i], contient tous les schémas maximaux (au sens de l'inclusion ensembliste) S pour lesquels il existe $j < i$ tels que $t_j.S = t_i.S$.

Donc, quand nous considérons t_i durant le `scan` de r, sachant que S est dans AUX(r)[i] on est sûr que pour tout $X \subseteq S$, $t_i.X$ a déjà été rencontré.
L'algorithme correspondant permettant de calculer la table AUX est représenté par l'Algorithme 15, avec $match(t_i, t_j)$ représentant l'ensemble des attributs A tels que $t_i.A = t_j.A$.

Exemple 3.9. *Nous illustrons la construction de la table $AUX(J)$ en utilisant notre base de données de référence représenté par la Figure 3.1, pour $J = Cust \bowtie Prod \bowtie Sales$. Les tables J et $AUX(J)$ sont représentées par la Figure 3.2.*
Comme $match(t_2, t_1) = Caddr$, nous obtenons $AUX(J)[2] = Caddr$. De manière similaire, comme $match(t_3, t_1) = \{Pid, Caddr, Ptype\}$ et $match(t_3, t_2) = \{Cid, Cname, Caddr\}$, $AUX(J)[3]$ est l'ensemble de ces deux ensembles d'attributs, puisque aucun des deux n'est inclus dans l'autre.
Le calcul de $AUX(J)[4]$ est similaire, et bien que $match(t_4, t_3) = Caddr$, ce schéma n'apparait pas dans $AUX(J)[4]$, ceci est dû au fait que $Caddr$ est un sous ensemble de $\{Cid\, Cname\, Caddr\, Qty\}$ et $\{Pid\, Caddr\, Ptype\}$ qui sont dans $AUX(J)[4]$.

Soient la table r et la table AUX(r) déja calculée, les supports des classes d'équivalence sur r sont calculés en parcourant parallélement r et AUX(r). L'algorithme correspond `scan` par l'Algorithme 14. L'entrée de l'algorithme `scan` est l'ensemble C des classes génériques candidates de la forme $\langle X, Y, r\rangle$ où r contenant les tuples $t_1, \ldots, t_n$.

L'ensemble des classes candidates fréquentes associées au classes générique candidates de C sont calculées de la manière suivante : Pour tout $i = 1, \ldots, n$, les actions suivantes sont effectuées, pour chaque $\langle X, Y, r\rangle$ de C :

1. Si AUX(r)[i] contient un sur schéma de X, alors $t_i.X$ a déjà été rencontré pour un j tel que $j < i$. Ainsi $t_i.X$ a déjà été rencontré par une projection de X, sinon, $t_i.X$ est rencontré pour la première fois.
2. Sinon, $t_i.X$ doit être compté pour le calcul du support de $q = \pi_X \sigma_{t_i.Y}(r)$. Deux cas sont possibles :

Algorithm 14 SCAN

Entrées: Une table de dimension r de la base de donnés ou la table de jointure ayant pour schéma R l'ensemble C des classes génériques candidates, la table AUX(r).

Sortie: L'ensemble L des classes génériques fréquentes de C, et les classes fréquentes associées $L_{Freq}(r)$.

1: $L = \emptyset$; $L_{Freq}(r) = \emptyset$
2: **Pour Tout** $\langle X, Y, r\rangle \in C$ **do**
3: $L(\langle X, Y, r\rangle) = \emptyset$
4: **Fin Pour Tout**
5: **Pour Tout** $i = 1, \ldots, n$ **do**
6: //r contient les tuples $t_1, \ldots, t_n$
7: **Pour Tout** $\langle X, Y, r\rangle \in C$ **do**
8: **Si** $\exists\ X' \in$ AUX(r)[i] tel que $X \subseteq X'$ **Alors**
9: //$t_i.X$ a été déja rencontré
10: `On ne fait rien`
11: **Sinon**
12: **Si** $\exists\ Y' \in$ AUX(r)[i] tel que $Y \subseteq Y'$ **Alors**
13: //$\pi_X \sigma_{t_i.Y}(r)$
14: //$t_i.X$ doit être pris en compte pour le calcul du support de $\pi_X \sigma_{t_i.Y}(r)$
15: $sup(\pi_X \sigma_{t_i.Y}(r)) = sup(\pi_X \sigma_{t_i.Y}(r)) + 1$
16: **Sinon**
17: //$\pi_X \sigma_{t_i.Y}(r)$ n'a pas encore été rencontré
18: // Soit on élague la requête soit on initialise son support à 1.
19: **Si** not(pruneQuery($\pi_X \sigma_{t_i.Y}(r)$)) **Alors**
20: $sup(\pi_X \sigma_{t_i.Y}(r)) = 1$
21: $L(\langle X, Y, r\rangle) = L(\langle X, Y, r\rangle) \cup \{\pi_X \sigma_{t_i.Y}(r)\}$
22: **Fin Si**
23: **Fin Si**
24: **Fin Si**
25: **Fin Pour Tout**
26: **Fin Pour Tout**
27: **Pour Tout** $\langle X, Y, r\rangle \in C$ **do**
28: $L(\langle X, Y, r\rangle) = L(\langle X, Y r\rangle) \setminus \{\pi_X \sigma_y(r) \mid sup(\pi_X \sigma_y(r)) < min\text{-}sup\}$
29: **Si** $L(\langle X, Y, r\rangle) \neq \emptyset$ **Alors**
30: $L_{Freq}(r) = L_{Freq}(r) \cup L(\langle X, Y, r\rangle)$
31: $L = L \cup \{\langle X, Y, r\rangle\}$
32: **Fin Si**
33: **Fin Pour Tout**
34: **Retourner** L and $L_{Freq}(r)$

Algorithm 15 AUX

Entrées: Une table r contenant les tuples $t_1, \ldots, t_n$.
Sortie: La table AUX(r).

1: AUX[1] = $\emptyset$
2: **Pour Tout** $i = 2, \ldots, n$ **do**
3: AUX(r)[i] = $\emptyset$
4: **Pour Tout** $j = 1, \ldots, i-1$ **do**
5: compute $match(t_i, t_j)$
6: **Si** AUX(r)[i] ne contient pas un sur ensemble de $match(t_i, t_j)$ **Alors**
7: AUX(r)[i] = AUX(r)[i] $\cup$ $match(t_i, t_j)$
8: AUX(r)[i]=AUX(r)[i] $\setminus$ $\{Y | Y \in match(t_i, t_j)\}$
9: **Fin Si**
10: **Fin Pour Tout**
11: **Fin Pour Tout**
12: **Retourner** AUX(r)

J	Cid	Pid	$Cname$	$Caddr$	$Ptype$	Qty
t_1	c_1	p_1	John	Paris	milk	10
t_2	c_2	p_2	Mary	Paris	beer	5
t_3	c_2	p_1	Mary	Paris	milk	1
t_4	c_1	p_2	John	Paris	beer	10

AUX(J)	i	AUX(J)[i] ($1 \leq i \leq 4$)
	1	$\emptyset$
	2	$Caddr$
	3	($Pid\,Caddr\,Ptype$), ($Cid\,Cname\,Caddr$)
	4	($Cid\,Cname\,Caddr\,Qty$), ($Pid\,Caddr\,Ptype$)

FIGURE 3.2 – La table J et la table auxiliaire associée AUX(J) en considérant l'Exemple 3.1

(a) Si AUX(r)[i] contient un sur schéma de Y alors q a déjà été traité précédemment, et elle est associée à la classe générique $\langle X, Y, r\rangle$. Dans ce cas, le support de q est incrémenté.

(b) Autrement, q est traitée pour la première fois, ainsi, elle n'est pas associée à la classe générique $\langle X, Y, r\rangle$. Dans ce, nous regardons d'abord si q peut être élagué si oui nous l'élaguons

, sinon, son support est initialisé à 1 et nous associons q à $\langle X, Y, r\rangle$.

Une fois cette action terminée, les supports de toutes les classes sont connus. Toutes les classes telles que leur support est supérieur ou égal à *min-sup* sont mises dans $L_{Freq}(r)$ et cet ensemble est retourné.

L'exemple suivant illustre notre méthode de calcul des supports en utilisant la table AUX :

Exemple 3.10. *Soit l'instance de base de données de la Figure 3.1, et reprenons la table AUX de la table de jointure J calculée précédemment et représentée par la Figure 3.2. Evaluons le support de la requête* $q = \pi_{Pid\,Ptype\,Caddr}\sigma_{\texttt{Paris}}(J)$ *:*

- *Initialement,* $sup(q) = 0$
- $AUX[1] = \emptyset$ $(\nexists X' \in AUX[1](J) : X \subseteq X')$ *et* $t_i.Caddr =$ `Paris` *alors* $sup(q) = 0 \rightarrow sup(q) = 1$
- $AUX[2] = \{Caddr\}$ $(\nexists X' \in AUX[2](J) : X \subseteq X')$ *et* $t_i.Caddr =$ `Paris` *alors* $sup(q) = 1 \rightarrow sup(q) = 2$
- $AUX[3] = \{(Pid\,Caddr\,Ptype), (Cid\,Cname\,Caddr)\}$ $(\exists X' \in AUX[3](J) : X \subseteq X')$ *alors* rien à faire.
- $AUX[4] = \{(Cid\,Cname\,Caddr\,Qty), (Pid\,Caddr\,Ptype)\}$ $(\exists X' \in AUX[4](J) : X \subseteq X')$ *alors* rien à faire.

Nous obtenons, finalement, $sup(q) = 2$

3.5 Complexité de FQF

Comme pour la plupart des algorithmes par niveau, la complexité de notre algorithme sera exprimée en fonction du nombre de passes sur la base de données Δ, car, comme mentionné précédemment, le support de toutes les classes candidates de la forme $\pi_X\sigma_y(r)$ d'un niveau donné peut être obtenu en une seule passe sur r.
Ainsi, la complexité de notre algorithme est donnée par le nombre d'itérations effectué lors de la recherche des classes fréquentes de la table r.
Ce nombre d'itérations est égal à la longueur S de l'une des plus longues séquences des classes de requêtes $(q_1, ..., q_S)$ telles que q_1 et q_S sont respectivement la classe générique la moins spécifique et la classe générique la plus spécifique de $\mathcal{C}^*$ de la forme $\pi_X\sigma_y(r)$, et $q_{i+1} \in succ(q_i)$ pour $i = 1, ..., S-1$.
En considérant que la table r à parcourir est définie sur le schéma R, montrons que S est linéaire par rapport à $|R|$ ($\mathcal{O}(|R|)$).
La Proposition 3.6 montre que pour toutes classes $q = \pi_X\sigma_y(r)$ et $q = \pi_{X'}\sigma_{y'}(r)$ telles que $q' \in succ(q)$ et $q \neq C_1$, nous avons soit $|X' \setminus Y'| = |X \setminus Y|$ (si q' est obtenue grâce à la Proposition 3.6(4)), où $X' = XA$ et $Y' = YA$), soit $|X' \setminus Y'| = |X \setminus Y| - 1$ (si q' est obtenue grâce à la Proposition 3.6(2), où $X' = X \setminus A$ et $Y' = Y$, ou grâce à la Proposition 3.6(3) avec $X' = X$ et $Y' = YA$).

De plus, pour $i = 1, \ldots, S$, soit $q_i = \pi_{X_i}\sigma_{y_i}(r)$, notons par d_i la cardinalité de $(X_i \setminus Y_i)$. Comme $\langle R, \top, r\rangle$ et C_1 sont respectivement la classe la moins spécifique et la classe la plus spécifique de $\mathcal{C}$, nous avons $d_1 = |R|$ et $d_S = 0$ (comme toutes les requêtes $\pi_X\sigma_x(r)$ avec $X \in cl(F)$ sont dans C_1 et en particulier lorsque $X = \emptyset$). Ainsi, $(d_1, \ldots, d_S)$ est une séquence décroissante d'entiers allant de $|R|$ à 0, dans laquelle au plus $|R|$ termes sont égaux (à cause du fait que la Proposition 3.6(4) ne peut pas être appliquée deux fois avec le même attribut A dans la séquence comme $A \in R \setminus X$). Donc, nous avons $S \leq 2.|R| + 1$.
D'autre part, soit $(q_1, \ldots, q_s)$ une séquence construite de la manière suivante, en commençant avec $q_1 = \langle R, \top, r\rangle$:

1. pour atteindre une classe de la forme $\langle A, \top, r\rangle$, où $A \in (D_{i_0} \setminus K_{i_0})$ $(i_0 \in \{1, \ldots, N\})$ il faut enlever de X_i, un par un, tous les attributs clés de K puis, tous les attributs

restants sauf A. Cette étape génère $|R|-1$ classes telles que $q_{i+1} \in succ(q_i)$ (grâce à la Proposition 3.6(2)), ainsi nous obtenons une séquence de longueur $|R|$.

2. Ensuite, étendre le schéma de projection et de sélection en ajoutant, un à un, à X_i et Y_i tous les attributs différents de A et K_{i_0} (En considérant en premier les attributs non clés, puis les attributs clés).

   Cette étape génère $|R|-2$ classes telles que $q_{i+1} \in succ(q_i)$ (grâce à la Proposition 3.6(4)) et $q_{2.(|R|-1)} = \pi_{R \setminus K_{i_0}} \sigma_y(r)$ où y est défini sur $U \setminus (K_{i_0} A)$.

   Comme $succ(q_{2.(|R|-1)}) = C_1$ (grâce à la Proposition 3.6(1)), $s = 2.|R| - 1$.

Ainsi, nous avons, $2.|R|-1 \leq S \leq 2.|R|+1$. Ce qui implique que *la complexité du nombre de passes de notre algorithme sur la table r est en $\mathcal{O}(|R|)$*.
En considérant, l'Algorithme 9, les tables à considérer sont les tables de dimension (δ_i) pour $i \in \{1..N\}$ et la table de jointure J, la complexité globale de l'algorithme en fonction du nombre de passes est alors $\mathcal{O}(|D_1| + |D_2| + ... + |D_N| + |U|)$ qui est en $\mathcal{O}(N.|U|)$.
Ainsi, nous en déduisons que la complexité globale de notre algorithme en fonction du nombre de passes est linéaire en fonction de la taille de l'univers U de la base de données Δ.
Il faut noter cependant que le calcul de la table $AUX(r)$ est quadratique en fonction de la taille de r cependant, il faut remarquer que :

1. Le calcul de table auxiliaire peut être vu comme un "pre-processing", car cette table est calculée seulement *une fois* pour toutes les exécutions de FQF, en supposant bien sûr que la base de données n'est pas mise à jour.

   Cette remarque est très importante par rapport au temps d'exécution, comme nous le verrons dans le chapitre implémentation (Figure 4.9), lorsque la table auxiliaire est disponible, le temps d'exécution de FQF devient très faible.

2. Lorsque la table r de la base de données est mise à jour, la mise à jour de la table AUX(r) peut être effectuées efficacement.

   En effet, dans le cas de l'insertion d'un nouveau tuple t dans r, et en supposant que t est le dernier tuple de r, une nouvelle ligne est ajoutée à AUX(r) et les schémas associés sont obtenus en une seule passe sur r. Si un tuple t_i est supprimé, alors on supprime tout simplement AUX(r)[t_i] de AUX(r), et seules les lignes AUX(r)[t_j] telles que $j > i$ et $match(t_j, t_i) \in$ AUX(r)[t_j] devront être mises à jour.

3.6 Une approche utilisant les résultats de notre contribution : Conqueror$^+$ ([99])

Dans notre état de l'art, nous avons mentionné l'existence d'une approche appelée Conqueror$^+$ utilisant les idées de notre contribution afin d'améliorer les performances de l'approche Conqueror ([56]) présenté dans le chapitre précédent. Cette approche consiste à utiliser les dépendances fonctionnelles et d'inclusion présentes dans la base de données, pour améliorer la génération des requêtes candidates et élaguer les requêtes redondantes. Elle permet également de découvrir, durant l'exécution de l'algorithme, de nouvelles dépendances fonctionnelles et d'inclusion inconnues à priori. Ces nouvelles dépendances

fonctionnelles et d'inclusion découvertes sont ensuite aussitôt utilisées pour élaguer la liste des classes candidates de l'étape suivante.
En effet, les auteurs se sont rendus compte que dans `Conqueror` plusieurs requêtes candidates équivalentes relativement au dépendances fonctionnelles, donc ayant le même support, sont obtenues lors de la phase de génération. Les auteurs définissent alors une relation de pré-ordre sur les requêtes qui permettra d'optimiser la génération, en évitant de générer des requêtes équivalentes, mais également, d'élaguer l'ensemble des candidats. Cette relation de pré-ordre est basée sur les idées de notre approche.

Nous allons tout d'abord donner quelques définitions et propriétés de cette relation de pré-ordre, puis nous allons présenter l'algorithme `Conqueror`$^+$. Nous reprendrons les mêmes notations que celles utilisées dans la Section 2.6.3.
La relation de pré-ordre utilisée par `Conqueror`$^+$, que l'on notera $\preceq_c$ pour la différencier de celle que nous avons proposée précédemment, est donnée par la Définition 3.8. Nous rappelons que :

- $\bowtie$ (Q) représente la condition de jointure de Q,
- $\pi(Q)$ la projection de Q,
- Q^σ la conjonction de condition de sélection de Q,
- la requête de jointure $J(Q)$ est la requête telle que sa condition de sélection est vide, ses attributs de projection sont constitués de l'ensemble des attributs des relations apparaissant dans Q et telle que $\bowtie$ $(J(Q))$ =$\bowtie$ (Q),
- $blocs(Q)$ représente les blocs engendrés par $\bowtie$ (Q).

Ces différentes notations sont illustrées par l'Exemple 3.11.

Exemple 3.11. *Considérons la base de données $\mathcal{D}$ constituée de deux deux relations R_1 et R_2 ayant les schémas suivants : $sch(R_1) = \{A, B\}$ et $sch(R_2) = \{C, D, E\}$. En accord avec la Définition 3.8, R est le produit cartésien de R_1 et R_2, et si on considère la requête $Q_1 = \pi_{AD}\sigma_{(A=C)\wedge(E=e)}(R)$, $\bowtie$ $(Q) = \{A = C\}, \pi(Q_1) = \{A, D\}$, $Q_1^\sigma = \{E = e\}$ et $J(Q_1) = \pi_{ABCDE}\sigma_{\bowtie(Q_1)}(R)$, $blocs(Q_1) = \{\{A, C\}, \{B\}, \{D\}, \{E\}\}$.*

Définition 3.8. *Soient $Q_1 = \pi_{X_1}\sigma_{F_1}R$ et $Q_2 = \pi_{X_2}\sigma_{F_2}R$ deux requêtes conjonctives telles que définies dans `Conqueror`. Notons par Y_i le schéma de Q_i^σ, pour $i = 1, 2$, nous avons $Q_1 \preceq_c Q_2$ si et seulement si :*

1. *$\bowtie(Q_1) \subseteq \bowtie(Q_2)$,*
2. *$J(Q_2)(\mathcal{I})$ satisfait $X_1Y_2 \rightarrow X_2$ and $Y_2 \rightarrow Y_1$, et*
3. *le tuple $Q_1^\sigma Q_2^\sigma$ est dans $\pi_{Y_1Y_2}J(Q_2)(\mathcal{I})$.*

Exemple 3.12. *Considérons l'Exemple 3.11, et supposons que $FD_1 = \emptyset$ et $FD_2 = \{C \rightarrow D, E \rightarrow D\}$, et soit $Q_1 = \pi_{AD}\sigma_{(A=C)\wedge(E=e)}(R)$ et $Q_2 = \pi_C\sigma_{(A=C)\wedge(D=d)}(R)$. Nous avons : $\bowtie$ (Q_1) =$\bowtie$ (Q_2) et $J(Q_1) = J(Q_2) = \pi_{ABCDE}(R)$. Ainsi, si $\mathcal{I}$ est l'instance de $\mathcal{D}$, $J(Q_2)(I)$ satisfait FD. De plus à cause de l'égalité défini par $\bowtie$ (Q_2), $J(Q_2)(I)$ satisfait également $A \rightarrow C$ et $C \rightarrow A$. Ainsi, $J(Q_2)(I)$ satisfait $CE \rightarrow AD$ et $E \rightarrow D$, si de plus, $de \in \pi_{DE}J(Q_2)(\mathcal{I})$ grâce à la Definition 3.8, nous déduisons que $Q_2 \preceq_c Q_1$.*

Les auteurs montrent dans [99] que $\preceq_c$ est une relation de pré-ordre sur l'ensemble des requêtes conjonctives de $\mathcal{D}$, mais aussi que la mesure de support est anti-monotone par

rapport à cette relation d'ordre, autrement dit pour tout Q_1 et Q_2 telles que $Q_1 \preceq_c Q_2$, nous avons $support(Q_2) \leq support(Q_1)$.
Ainsi, grâce à l'anti-monotonie, les auteurs proposent un algorithme par niveau pour la recherche des requêtes fréquentes de la même manière que dans Apriori ([8]) pour chacune des trois boucles de Projection, de Jointure et de Sélection que nous présenterons.
De plus, la relation de pré-ordre $\preceq_c$, ainsi définie, induit une relation d'équivalence que les auteurs notent $\sim$. Cette relation d'équivalence est définie de la manière suivante : soient deux requêtes conjonctives simples Q_1 et Q_2, $Q_1 \sim Q_2$ si et seulement si $Q_1 \preceq_c Q_2$ et $Q_2 \preceq_c Q_1$.
Grâce à l'anti-monotonie, du support par rapport à $\preceq_c$, si $Q_1 \sim Q_2$ alors $support(Q_1) = support(Q_2)$. Ainsi, un seul calcul par classe d'équivalence modulo $\sim$ permet d'avoir le support de toutes les requêtes appartenant à cette classe d'équivalence.
Nous rappelons que X^+ est la fermeture transitive de X relativement aux dépendances fonctionnelles connues FD.
Ainsi, en se basant sur notre contribution, nous pouvons voir que pour deux requêtes $Q_1 = \pi_{X_1}\sigma_{F_1}R$ et $Q_2 = \pi_{X_2}\sigma_{F_2}R$, nous avons $Q_1 \sim Q_2$ si et seulement si $\bowtie(Q_1) = \bowtie(Q_2)$, $(X_1Y_1)^+ = (X_2Y_2)^+$, $Y_1^+ = Y_2^+$ et $Q_1^\sigma Q_2^\sigma \in \pi_{Y_1Y_2}J(Q_1)(\mathcal{I})$.
Dans ce qui suit, les requêtes considérées par Conqueror$^+$, sont les représentants des classes d'équivalence définis de la manière suivante : soit une classe d'équivalence contenant une requête Q, le représentant de cette classe est noté Q^+ et est définie telle que $\pi(Q^+) = \pi(Q)^+$, $\bowtie(Q^+) = \bowtie(Q)$ et $\sigma(Q^+)$ est la condition de sélection correspondant au sur tuple de Q^σ, notée $(Q^\sigma)^+$, définie sur $sch(Q^\sigma)^+$, ayant pour domaine $\pi_{sch(Q^\sigma)^+}J(Q)(\mathcal{I})$.
De plus, si $\pi(Q) \subseteq sch(Q^\sigma)$ alors le support de Q est égal à 1, ce support est toujours inférieur au seuil minimum de support dans le cas pratique, ainsi les requêtes Q intéressantes sont telles que :

$$\pi(Q) = \pi(Q)^+,\ sch(Q^\sigma) = sch(Q^\sigma)^+, \text{ et } sch(Q^\sigma) \subset \pi(Q).$$

Exemple 3.13. *En reprenant les requêtes Q_1 et Q_2 de l'Exemple 3.12, nous pouvons remarquer que ces deux requêtes ne satisfont pas les conditions précédentes. En effet, comme $sch(Q_1^\sigma) = E$ et $\pi(Q_1) = AD$, l'inclusion $sch(Q_1^\sigma) \subset \pi(Q_1)$ n'est pas satisfaite. On peut vérifier également qu'aucune de ces deux requêtes n'est fermée, ainsi ni l'une ni l'autre ne sera considérée dans l'Algorithme* `Conqueror`$^+$. *Mais, comme $J(Q_1)(\mathcal{I})$ satisfait $C \to D, E \to D, A \to C$ et $C \to A$,* `Conqueror`$^+$ *considèrera en lieu et place les requêtes Q_1^+ et Q_2^+ définies de la manière suivante :*
- $Q_1^+ = \pi_{ACDE}\sigma_{(A=C)\wedge(E=e)}(R)$,
- $Q_2^+ = \pi_{ACDE}\sigma_{(A=C)\wedge(E=e)\wedge(D=d)}(R)$.

Il faut noter que le fait de considérer de telles requêtes réduit considérablement le nombre de requêtes fréquentes retournées car si le représentant d'une classe d'équivalence de requêtes est fréquent alors tous les requêtes qui sont dans cette classe d'équivalence sont également fréquentes.
Dans ce qui suit, nous allons présenter l'algorithme `Conqueror`$^+$ qui est une amélioration de l'Algorithme `Conqueror` dans la mesure où il prend en compte les dépendances fonctionnelles et d'inclusion, mais également, il permet de découvrir de nouvelles dépendances fonctionnelles.

Nous rappelons que l'algorithme Conqueror$^+$ considère les classes d'équivalence à la place des requêtes et chaque classe d'équivalence est représentée par un représentant tel que définie précédemment. De plus, chaque requête Q entretient une liste de dépendances fonctionnelles vérifiées par $J(Q)(\mathcal{I})$, cette liste est ainsi mise à jour au fur et à mesure de la découverte de nouvelles dépendances fonctionnelles au cours de l'algorithme. Cette liste servira à élaguer l'espace de recherche, comme nous le verrons dans la suite.

L'Algorithme Conqueror$^+$, comme Conqueror, est constitué de trois boucles, dans ce qui suit, nous allons présenter ces différentes boucles mais surtout leurs différences majeures par rapport à celles de Conqueror.

3.6.1 Boucle de Jointure :

Cette boucle consiste à générer l'ensemble des partitions possibles des attributs de jointure en utilisant une stratégie en largeur d'abord. Cette génération est effectuée grâce au "Restricted growth string" (chaque "restricted growth string" représentant une partition de l'ensemble des attributs comme dans Conqueror). Par exemple en reprenant l'Exemple 3.11, l'ensemble U des attributs de $\mathcal{D}$ est $\{A, B, C, D, E\}$. Par exemple, la "restricted growth string" 12231 représente la condition de sélection $(A = E) \wedge (B = C)$, ce qui correspond à la partition $\{\{A, E\}, \{B, C\}, \{D\}\}$. Egalement, comme dans Conqueror, cette boucle teste également si la requête générée est plus générale que les requêtes les plus spécifiques définies par l'utilisateur, si celle ci est plus spécifique, elle est ignorée. De plus, la requête générée hérite des dépendances fonctionnelles des ancêtres mais aussi si JQ est la requête obtenue par jointure à partir de Q, JQ hérite des dépendances de fonctionnelles obtenues à partir de la jointure de deux attributs. En effet, si deux attributs sont tels que $A = B$, nous avons les dépendances fonctionnelles $A \rightarrow B$ et $B \rightarrow A$.

3.6.2 Boucle de Projection

Pour chaque classe de requêtes Q générée à l'étape précédente, générer l'ensemble des projections possibles PQ, en supprimant un bloc, parmi les blocs engendrés par $\bowtie (Q)$, du schéma de projection de Q comme dans Conqueror (voir Exemple 3.14). Cependant, puisque nous travaillons qu'avec des requêtes dont la projection est fermée, il est possible que la suppression d'un bloc d'attributs au niveau de la projection engendre un schéma non fermé, c'est pourquoi les auteurs vérifient, avant d'évaluer le support, d'une requête si le schéma de projection obtenue est fermée ou pas, si elle n'est pas close la requête générée est tout simplement mise en queue dans la liste des candidats obtenus par projection sans traitements supplémentaires et si le schéma est fermé, le support par rapport à l'instance de base de données $\mathcal{I}$ est évalué et la requête est mise dans la liste des fréquents si son support dépasse le seuil minimal de support *min-sup*.
De plus, durant cette boucle, de nouvelles dépendances fonctionnelles sont découvertes grâce à la Remarque 3.3.

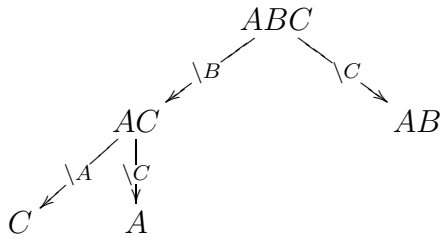
Remarque 3.3. *Soient une relation r défini sur un schéma R, et deux schémas X et X' inclus dans R tels que $X \subseteq X'$: si $sup(\pi_X \sigma_y(r)) = sup(\pi_{X'} \sigma_y(r))$ alors nous avons la dépendance fonctionnelle :$X \rightarrow X'$ dans r.*

Les auteurs proposent alors de tester pour toute requête PPQ de la liste des classes fréquentes notée FPQ, telle qu'il n'existe pas de PPQ' vérifiant $\pi(PQ) \subset \pi(PPQ') \subset \pi(PPQ)$ et si $support(PQ) = support(PPQ)$ alors la dépendances fonctionnelles $\pi(PQ) \rightarrow \pi(PPQ) \setminus \pi(PQ)$ est ajoutée à la liste des dépendances fonctionnelles vérifié par $J(Q)(\mathcal{I})$ et la requête PQ est marquée car elle est redondante. Cette liste permettra, comme nous le verrons dans le paragraphe suivant, d'élaguer la liste des requêtes candidates.

Naturellement, le fait de considérer que les requêtes dont les projections sont fermées crée des complications lors de la phase d'élagage. Pour contourner cette difficultés les auteurs proposent la stratégie suivante : soit PQ une requête fermée donnée, pour tous les prédécesseurs PPQ de PQ, la fermeture de PPQ par rapport à la liste des dépendances fonctionnelles vérifiée par $J(Q)(\mathcal{I})$ est calculée, avec Q représentant l'ancêtre de PQ. Si une des requêtes obtenues, c'est à dire les requêtes ayant pour projection la fermeture de PPQ, la même jointure et la même sélection que PQ n'est pas fréquente alors, grâce à la propriété d'anti-monotonie, la requête PQ est élaguée.

Une autre différence majeure de la boucle de projection de `Conqueror`$^+$ par rapport à celle `Conqueror` est le fait que les sélections sont effectuées après la génération de toutes les projections, ce qui permet d'utiliser aussitôt toutes les dépendances fonctionnelles découvertes à une étape donnée pour élaguer la liste des requêtes /andidates

Exemple 3.14. *Soient 3 attributs A, B, C d'une relation r tels que $B \rightarrow A$, l'arbre de génération des projections par C est représenté par l'arbre suivant :*



Nous remarquons que le schéma de projection BC n'est pas généré puisque les schémas de projection doivent être fermés.

3.6.3 Boucle de Sélection :

Il est tout d'abord à remarquer que les requêtes marquées à l'étape précédente ne sont pas considérées, car ce sont des requêtes redondantes.

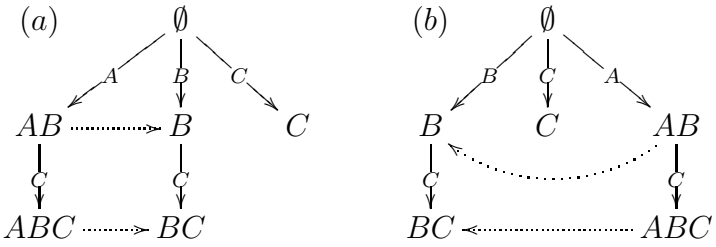
Dans cette boucle, les blocs d'attributs de projection $\pi(Q)$ qui ne sont pas dans $\sigma(Q)$ de chaque requête Q obtenue à l'étape suivante sont ajoutés et instanciés un à un par des constantes de $\mathcal{I}$, en utilisant une stratégie en largeur d'abord. Finalement, les requêtes obtenues sont placées dans la liste des classes de requêtes fréquentes si leur support est supérieur au seuil minimal de support *min-sup*. Ici également, les blocs sont ordonnés pour ne pas générer des requêtes redondantes.

Cependant, la restriction selon laquelle, le schéma de sélection doit être fermé requiert la réorganisation les classes candidates pour pouvoir utiliser la propriété d'anti-montonie pour élaguer l'ensemble des requêtes candidates. L'Exemple 3.15 illustre ce point.

Exemple 3.15. *Soient les attributs A, B et C, et la dépendance fonctionnelle $A \rightarrow B$. La génération des ensembles de sélection. est représenté ci-dessous (a). En effet, l'addition de A au niveau du schéma de sélection entraine également l'ajout B pour que le schéma soit fermé.*

Cependant, à cause de la propriété d'anti-monotonie, il faut considérer B avant AB (car la sélection sur B est moins restrictive que la sélection sur AB). Les auteurs proposent alors de ré-ordonner la liste des classes candidates, pour être sûr que B est considéré avant AB et BC avant ABC, comme le montre l'arbre de génération (b) ci dessous.

□



3.6.4 Résultats expérimentaux :

L'efficacité de cette version de `Conqueror`, utilisant les idées de notre contribution, a été démontrée par ses auteurs grâce au développement du système *Conqueror*$^+$ ainsi qu'aux expérimentations comparatives menées.
Nous reviendrons sur les résultats comparatifs de `Conqueror` et *Conqueror*$^+$ dans le chapitre implémentation.

3.7 Conclusion

La recherche de requêtes fréquentes dans les bases de données relationnelles est un problème difficile.
Nous avons proposé une méthode, basée sur les dépendances fonctionnelles et les dépendances d'inclusion, qui permet de résoudre de manière efficace le problème dans le cas des bases de données ayant un schéma étoile.
En effet, une relation de pré-ordre qui utilise les dépendances fonctionnelles et les dépendances d'inclusion a été proposéee.
De plus, nous avons montré que le support est anti-monotone par rapport à cette relation de pré-ordre. L'anti-monotonie du support par rapport à la relation de pré-ordre nous permet d'utiliser une algorithme par niveau de type Apriori ([8]). De plus, cette relation de pré-ordre nous permet de définir des classes d'équivalence sur l'ensemble des requêtes. La définition de classes d'équivalence nous permet de ne considérer que les classes d'équivalence plutôt que les requêtes prises individuellement car les requêtes appartenant à une même classe d'équivalence ont même support.
Un algorithme efficace baptisé FQF qui permet de résoudre le problème a été proposé.
Nous avons également proposé une méthode efficace, basée sur une table auxiliaire (AUX) , qui permet d'évaluer efficacement le support des classes de requêtes considérées.

Une approches qui utilisent les résultats de nos travaux pour optimiser l'extraction des requêtes fréquentes a été présenté.
Notre algorithme a été implémenté, le chapitre suivant présente cette implémentation ainsi que les résultats de nos tests sur différentes bases de données synthétiques et réelles ayant un schéma étoile.
Nous présenterons également des résultats comparatifs de notre algorithme `FQF` par rapport à `Conqueror` ([56]) et `Conqueror`$^+$ ([99]).

4
IMPLÉMENTATION

4.1 Introduction

Le but de ce chapitre est de présenter, l'implémentation des algorithmes présentés dans cette thèse.

Notre implémentation a pour but, de tester le comportement des algorithmes sur des bases de données synthétiques ou réelles ayant un schéma étoile.

Ce chapitre est constitué de sept sections : la première section est l'introduction, la section 4.2 présentera l'interface graphique de notre application. La section 4.3 présentera les phases d'analyse, de conception et de développement. La section 4.4 présentera les classes de notre application. La section 4.5 présentera les difficultés rencontrées lors de la phase de développement et de test. La section 4.6 présentera quelques résultats expérimentaux de notre algorithme, mais également des résultats comparatifs avec les implémentations de `Conqueror` ([56]) et `Conqueror`$^+$ ([99]). Finalement, la dernière section fera la conclusion de ce chapitre.

4.2 Interface Graphique de l'Application

La figure 4.1 montre l'interface graphique de notre application, elle est composée des champs et boutons ci-dessous :

- ***Server Host :*** permet de donner l'adresse du serveur de base de données sur lequel on souhaite se connecter ;
- ***Database :*** permet de sélectionner la base sur laquelle on souhaite effectuer l'extraction ;
- ***Threshold :*** champs permet de rentrer le seuil minimal de support ;
- ***Username :*** permet à l'utilisateur d'entrer le login de connexion à la base ;
- ***Password :*** permet à l'utilisateur d'entrer le mot de passe de connexion à la base ;
- ***Start :*** ce bouton permet de démarrer la recherche des requêtes fréquentes ;
- ***Stop :*** Ce bouton permet d'arrêter la recherche.

FIGURE 4.1 – Interface Graphique de FQF

4.3 Analyse et Conception

4.3.1 Architecture générale de l'implémentation

La figure 4.2 illustre l'architecture globale de notre application.
L'utilisateur lance l'application à travers l'interface graphique présentée précédemment. Cette interface graphique lui permet de choisir, entre autres, le serveur de base de données et la base de données de travail, ces informations sont envoyées au moteur d'extraction qui se charge de la connexion et de la communication avec la base de données considérée. Le moteur d'extraction implémente l'algorithme de recherche de requêtes fréquentes proposé dans cette thèse.
Les requêtes fréquentes (avec leur support) découvertes sont sauvegardées, au fur et à mesure, dans un fichier texte.

4.3.2 Choix de langage et de Système de Gestion de base de données

Nous avons choisi d'effectuer notre implémentation, en utilisant des outils Free et Open Source, mais également de rendre le logiciel final aussi modulaire que possible, afin d'en faciliter la maintenance et son utilisation pour d'autres problèmes similaires au nôtre.

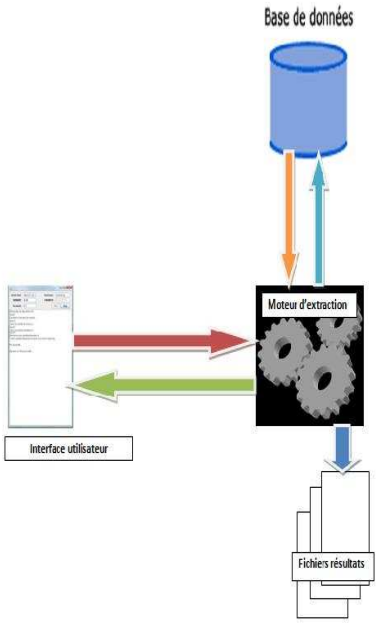


FIGURE 4.2 – Architecture globale de notre application de recherche de requêtes fréquentes (FQF)

De plus, la gestion de la mémoire étant un problème crucial et afin d'éviter les fuites de mémoire nous avons choisi d'implémenter notre application en *Java* qui est un langage qui élimine automatiquent les fuites de mémoire grâce au Garbage Collector , malgré le fait que les programmes écrit en *C++* soient plus rapides que les programmes écrits en *Java*.

4.4 Classes

Dans notre implémentation, nous avons besoin de stocker en mémoire centrale différents types de structures complexes comme les attributs, les schémas, les classes génériques, les requêtes ...

La taille de l'espace de recherche étant très importante, il peut arriver que nous ayons besoin de stocker à une itération donnée de l'algorithme un nombre très élevé d'objets ayant de telles structures. Il devient alors nécessaire de définir efficacement ces structures afin d'utiliser le moins d'octets possibles pour éviter la saturation de la mémoire, il est également nécessaire d'utiliser des références entre les objets pour éviter des duplications d'objets.

Pour plus d'efficacité, nous utiliserons également les classes prédéfinies de la bibliothèque de *Java* comme les collections (Vector, Set, HashSet, HashMap, ...) pour manipuler efficacement l'ensemble des classes génériques candidates, l'ensemble des classes génériques fréquentes mais aussi, les requêtes fréquentes à une itération donnée de l'algorithme.

Pour les besoins de nos algorithmes, nous avons besoin de définir principalement les classes

suivantes :

- Attribute (attribut),
- Schema (schéma),
- GenericClass (classe générique),
- QueryClass (classe de requêtes).
- Tuple (tuple d'une table).
- Generate (classe permettant de générer les candidats)
- Prune (Classe permettant d'élaguer les classes candidates non fréquentes)
- Scan (Classe permettant d'évaluer les supports des classes candidates)
- Aux (Classe permettant de calculer la table auxiliaire d'une table r)

Avant de commencer le développement de notre algorithme il nous fallait tout d'abord créer un générateur de base de données étoiles pour nos tests. La section suivante présente brièvement notre générateur de base de données.

4.4.1 Générateur de base de données sous forme de schéma étoile

Nous avons eu à développer, pour les besoins de nos tests et de nos expériences, un générateur de bases de données de type étoile que nous avons appelé **SSG** (**S**tar **S**chema **G**enerator).
Cette application permet de générer des bases de données ayant un schéma étoile avec des caractéristiques paramétrables comme :

- le nombre de dimensions,
- le nombre d'attributs par dimension,
- le nombre de mesures,
- le nombre de tuples par table de dimension et le nombre de tuples dans la table de faits.

Notre générateur est une extension du générateur proposé par [8] aux bases de données ayant un schéma étoile.

4.4.2 Développements

Après les phases d'analyse, de conception et de développement de l'application de génération des bases de données de type étoile, nous avons commencé le développement des méthodes des différentes classes ci-dessous en utilisant l'***IDE Eclipse Ganymede*** couplé au SGBD ***MySQL***.
C'est ainsi que nous avons implémenté un ensemble de classes que nous présenterons dans ce qui suit (pour plus de détails sur ces différentes classes le lecteur pourra consulter l'Annexe B).

- `Loader:` Cette classe contient les attributs et les méthodes permettant de se connecter par jdbc à notre base de données mysql.
  Elle permet également de générer à la volée la table de jointure J des différentes table de dimensions et la table de faits, mais également de parcourir un à un les tuples de la base.
  Elle contient les attributs :
  - url : l'url du serveur,

  - login : le login de connexion au serveur de base de données,
  - password : le mot de passe de connexion au serveur de base de données.
- `Tuple:` Cette classe représente la notion de tuple telle que définie dans cette thèse. Elle contient l'attribut :
  - mapAttToValues qui est un HashMap (associant un attribut à sa valeur).
- `Aux:` Cette classe contient les méthodes permettant de calculer la table AUX d'une table r telle que définie dans cette thèse :
  Elle contient les attributs :
  - url : l'url du serveur,
  - login : le login de connexion au serveur de base de données,
  - password : le mot de passe de connexion au serveur de base de données,
  - tableid : l'identifiant de la table dont on veut calculer la table AUX.
- `Saver:` Cette classe permet de sauvegarder dans un fichier les requêtes fréquentes d'une table r obtenues ainsi que leur support dans un fichier pour leur exploitation future.
  Elle contient l'attribut :
  - OUT_FILE_NAME_TXT qui représente le nom du fichier de sauvegarde des requêtes.
- `Schema:` Cette classe représente la notion de schéma telle que définie dans cette thèse. La classe Schema représente un ensemble d'attributs, `Schema` est représenté par un tableau de bits dans lequel chaque bit i représente un attribut A_i de U tels que si le bit i est à 1 l'attribut A_i est présent dans le schéma sinon l'attribut A_i est absent du schéma (comme le montre l'Exemple 4.1). Ainsi, la taille en octets d'un schéma est au maximum $|U|/8$ octets, ce qui permet d'utiliser le minimum d'espace mémoire possible.
  Cette classe contient l'attribut :
  - setOfAttribute qui est un tableau de bits, chaque bit correspond à un attribut si le bit i est à 1 l'attribut A_i est présent dans le schéma sinon l'attribut A_i est absent du schéma.

Exemple 4.1. *Considérons la table r contenant les 5 attributs suivants : A, B, C, D, E pris dans cette ordre.*
Par exemple, la table de bits suivant représentera le schéma BD.

0	*1*	*0*	*1*	*0*

- `GenericClass:` Cette classe représente la notion de classe générique telle que présentée dans cette thèse.
  Cette classe contient les attributs :
  - xGenericClass : le schéma de projection de type `Schema`,
  - yGenericClass : le schéma de sélection de type `Schema`.
- `QueryClass:` Cette classe permet de représenter la notion de requête telle que présentée dans cette thèse.
  Elle contient les attributs :
  - xSchema : le schéma de projection de type `Schema`,
  - ySchema : le schéma de sélection de type `Schema`,

– yValue : le tuple correspondant à la sélection sur le schéma ySchema de type `Tuple`.

– `Frequent Queries`: Cette classe est la classe principale de notre application.

Le diagramme de classes et les différents diagrammes de séquences de l'application sont disponibles en Annexe C.

4.5 Problèmes rencontrés et Solutions proposées

4.5.1 Problèmes rencontrés

Lors de l'implémentation et des tests des algorithmes proposés, nous avons été confrontés à un certain nombre de difficultés qui sont dues :

– aux redondances lors de la phase de génération (nous rappelons que nous appelons redondance le fait que deux classes génériques d'un même niveau n génèrent le même noeud fils au niveau suivant $n+1$) ;

– à la taille de l'espace de recherche, nous avons souvent eu des dépassements de mémoire *(OutOfMemory Error en* java*)*, dès que le nombre d'attributs dépasse 12 lors des tests.

Ce dépassement de mémoire durant l'exécution de l'algorithme est dû d'une part aux redondances (le fait que deux noeuds différents génèrent le même noeud fils lors de la phase de génération) et d'autre part au nombre important de requêtes à sauvegarder en mémoire centrale à une itération donnée de l'algorithme.

Pour contourner ces deux difficultés, nous avons après analyse :

– éliminé partiellement les redondances ;

– implémenté les classes génériques avec le moins d'octets possibles (en représentant un schéma par un tableau de bits de taille maximale $|U|/8$ octets).

Dans ce qui suit, nous allons détailler notre stratégie d'élimination des redondances.

4.5.2 Différents cas de redondance

Dans cette section, nous allons donner, par des exemples, quelques différentes configurations pouvant mener à des redondances, puis nous présenterons des méthodes efficaces permettant de les éviter.

Ces différentes configurations de redondances recensées sont illustrées par l'Exemple 4.2

Exemple 4.2.

1. *Redondances liées à l'application de l'item 2 de la proposition 3.6 (Projection) :*

   Soit une table r ayant 5 attributs $K_1, K_2, D_{11}, D_{21}, M_1$ vérifiant les dépendances fonctionnelles suivantes : $K_1 \rightarrow D_{11}$, $K_2 \rightarrow D_{21}$ et $K_1K_2 \rightarrow M_1$. Ce type de redondance est illustré par la figure 4.3.

   Sur ce graphique, nous remarquons que les classes génériques $\langle K_1D_{11}D_{21}M_1, \emptyset, r\rangle$ et $\langle K_2D_{11}D_{21}M_1, \emptyset, r\rangle$ génèrent le même noeud successeur $\langle D_{11}D_{21}M_1, \emptyset, r\rangle$.

2. *Redondances liées à l'application de l'item 3 de la proposition 3.6 (Sélection) :*
   Soit une table r ayant les attributs $K_1, K_2, D_{11}, D_{21}, M_1$ vérifiant les dépendances fonctionnelles suivantes : $K_1 \rightarrow D_{11}$, $K_2 \rightarrow D_{21}$ et $K_1K_2 \rightarrow M_1$. Ce type de redondance est illustré par la figure 4.4. Nous remarquons que les classes génériques $\langle K_1K_2D_{11}D_{21}M_1, D_{11}, r\rangle$ et $\langle K_1K_2D_{11}D_{21}M_1, D_{21}, r\rangle$ génèrent le même noeud successeur $\langle K_1K_2D_{11}D_{21}M_1, D_{11}D_{21}, r\rangle$.

3. *Redondances liées à l'application de l'item 4 de la proposition 3.6 (Projection et Sélection) :*
   Soit une table r ayant trois attributs non clés D_{11}, D_{21} et D_{31}.
   Ce type de redondance est illustré par la figure 4.5. Sur cette figure, nous remarquons que les classes génériques $\langle D_{11}D_{31}, D_{11}, r\rangle$ et $\langle D_{21}D_{31}, D_{21}, r\rangle$ génèrent le même noeud successeur $\langle D_{11}D_{21}D_{31}, D_{11}D_{21}, r\rangle$.

4. *Redondances dues à l'application simultanée de proposition 3.6(3) puis proposition 3.6(2) :*
   Considèrons, une table r ayant cinq attributs $K_1K_2D_{11}D_{21}M_1$ tels que $K_1 \rightarrow D_{11}, K_2 \rightarrow D_{21}$ et $K_1K_2 \rightarrow M_1$, sur la figure 4.6, nous remarquons que la classe générique $\langle K_1K_2D_{11}D_{21}M_1, D_{21}, r\rangle$ obtenue par l'application de la proposition 3.6(2) puis de la proposition 3.6(3) sur $\langle K_1K_2D_{11}D_{21}M_1, \emptyset, r\rangle$ est également obtenue par l'application de la proposition 3.6(3) puis de la proposition 3.6(2) sur $\langle K_1K_2D_{11}D_{21}M_1, \emptyset, r\rangle$. Ce type de redondance est illustré par la figure 4.6.
5. *Considérons, une table r ayant trois attributs K_1K_2 et M_1 tels que K_1K_2, vérifiant la dépendance fonctionnelle $K_1K_2 \rightarrow M_1$, sur la figure 4.7, nous remarquons que la classe générique $\langle K_2M_1, M_1, r\rangle$ obtenue par l'application de la proposition 3.6(3) puis de la proposition 3.6(2) sur $\langle K_1K_2M_1, \emptyset, r\rangle$ est également obtenue par l'application de la proposition 3.6(2) puis de la proposition 3.6(2) et enfin de la proposition 3.6(4) sur $\langle K_1K_2D_{11}D_{21}M_1, \emptyset, r\rangle$. Ce type de redondance est illustré par la figure 4.7.*

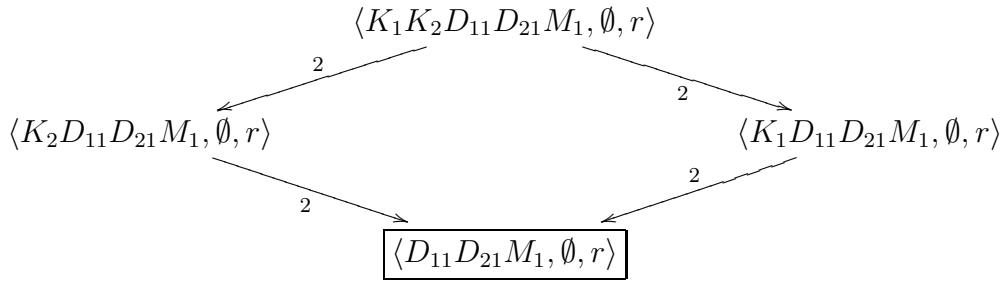


FIGURE 4.3 – Redondance de Projection (proposition 3.6(2))

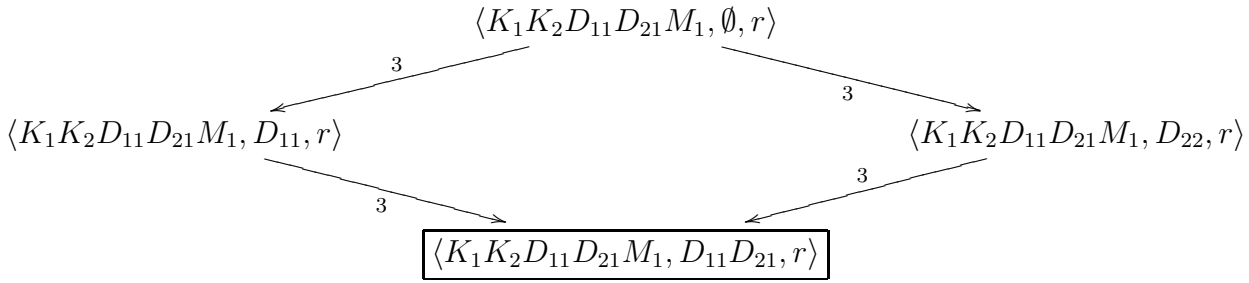


FIGURE 4.4 – Redondance de Sélection (proposition 3.6(3))

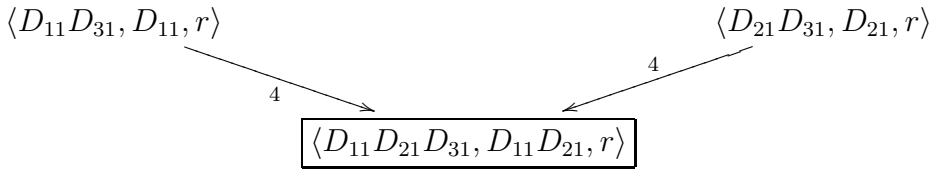


FIGURE 4.5 – Redondance de Projection-Sélection (proposition 3.6(4))

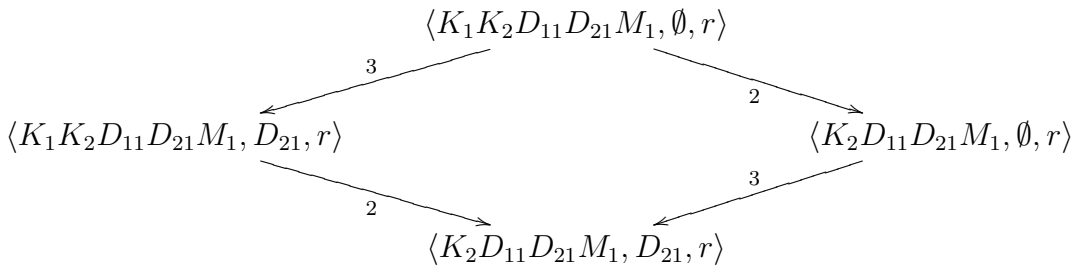


FIGURE 4.6 – Redondance induite par l'application de proposition 3.6(3) puis proposition 3.6(2) et proposition 3.6(2) puis proposition 3.6(3)

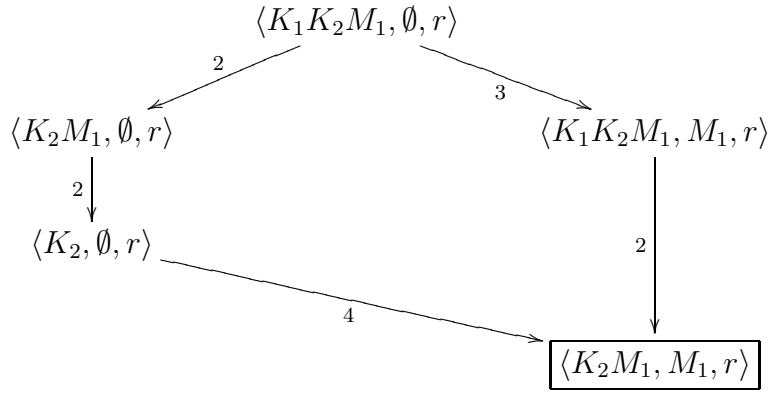


FIGURE 4.7 – Redondance induite par l'application de proposition 3.6(3) puis proposition 3.6(2) et proposition 3.6(2), proposition 3.6(2) puis proposition 3.6(4)

Nous avons énuméré ici les cas de redondances que nous avons recensés, cependant nous n'avons pas la preuve que d'autres cas de redondance n'existent pas.

4.5.3 Elimination partielle des redondances

Dans la suite, afin de présenter notre stratégie d'élimination des redondances, nous supposerons que les attributs sont ordonnés. L'ordre sur les attributs est donné par la définition 4.1 suivante.

Définition 4.1. *Soit une base de données définie sur un schéma étoile ayant pour univers* $U = K_1K_2...K_nD_{11}D_{12}\dots D_{1p_1}D_{21}D_{22}\dots D_{2p_2}\dots D_{22}...D_{2p_n}\dots M_1\dots M_m$, *où* K_i *représente la clé de la table de dimensions* δ_i, D_{ij} *le* j*-eme attribut de dimensions de la table* δ_i *et* M_i *la* i*-eme mesure de la table* φ, n *le nombre de dimension,* p_i *est le nombre d'attributs de dimensions de la table* δ_i *et* m *est le nombre de mesures. L'ordre d'un atttribut* A *noté* $\rho(A)$ *est défini de la manière suivante :*

- *si* $A = K_i$, $\rho(A) = i$;
- *si* $A = D_{ij}$, $\rho(A) = n + \sum_{k=1}^{i-1} p_k + j$.
- *si* $A = M_i$, $\rho(A) = |U| - m + i$.

De plus, comme nous l'avons mentionné précédemment, pour les besoins de notre implémentation et afin d'optimiser l'utilisation de la mémoire, nous représenterons chaque schéma $X \subseteq U$ par un tableau indicé de bits notée B_X qui est défini de la manière suivante :
Soit $X \subseteq U$, la chaine de bits B_X associée à X est définie de la manière suivante :

- le nombre de bits de B_X est égal à $|U|$;
- $\forall A \in U$, $B_X(\rho(A)) = 1$ si $A \in X$ et $B_X(\rho(A)) = 0$ si $A \notin X$.

Soit une classe générique C, la proposition suivante permet de générer des successeurs de C obtenus par application de la proposition 3.6(2) (Projection) sans redondance.

Proposition 4.1. *Soit* $C = \langle X, Y, r\rangle$ *une classe générique, et* $S_P(C)$ *l'ensemble des classes génériques défini par* $S_P(C) = S_P^1(C) \cup S_P^2(C) \cup S_P^3(C)$ *tel que, pour tout* $C' = \langle X', Y', r\rangle$ *de* S_P, *on a* $X' = X \setminus A$ *et* $Y' = Y$ *où* A *est un attribut de* X *qui satisfait l'une des trois conditions suivantes :*

1. $C' \in S_P^1(C)$ *si et seulement si* $\rho(A) \leq n$ (i.e., A *est une clé) et* $\rho(A) = \min_k\{k \mid (\forall l)(k < l \leq |U| \Rightarrow B_{X'}(l) = 1)\}$.
2. *Si* $C' \in S_P^2(C)$ *si et seulement si il existe* i *tel que* $B_X(K_i) = 0$, $\rho(D_{i1}) \leq \rho(A) \leq \rho(D_{ip_i})$ (i.e., $K_i \notin X$ *et* A *est un attribut de la dimension* i*) et* $\rho(A) = \min_k\{k \mid (\forall l)(k < l \leq |U| \Rightarrow B_{X'}(l) = 1)\}$.
3. *Si* $C' \in S_P^3(C)$ *si et seulement si il existe* i *tel que* $B_X(K_i) = 0$, $\rho(M_1) \leq \rho(A) \leq \rho(M_m)$ (i.e., $K_1 \dots K_n \not\subseteq X$ *et* A *est une mesure) et* $\rho(A) = \min_k\{k \mid (\forall l)(k < l \leq |U| \Rightarrow B_{X'}(l) = 1)\}$, *si* $B_{X'}(|U|) \neq 0$. *Sinon* $\rho(A) = |U|$.

Alors $S_P(C) \subseteq succ(C)$ et de plus, pour toute classe générique $C_1 = \langle X_1, Y_1, r\rangle$ distincte de C, on a $S_P(C) \cap S_P(C_1) = \emptyset$.

Démonstration. Il est facile de voir que, sous les hypothèses de la proposition, si X est fermé, alors X' est également fermé. Donc, $S_P(C) \subseteq succ(C)$ est une conséquence de la proposition 3.6(2).

Soit $C_1 = \langle X_1, Y_1, r\rangle$ une classe générique distincte de C telle que $S_P(C) \cap S_P(C_1) \neq \emptyset$, et soit $C' = \langle X', Y', r\rangle$ dans $S_P(C) \cap S_P(C_1)$. On a alors $Y_1 = Y' = Y$ (car tout C' de $S_P(C)$ est tel que $Y' = Y$) et donc, d'après la proposition 3.6, il existe A' tel que $X' = X \setminus A'$ (car $C' \in succ(C_1)$). Par suite, $B_{X_1}(\rho(A')) = 1$ et $B_{X'}(\rho(A')) = 0$ et pour tout autre attribut A'', $B_{X'}(\rho(A'')) = B_{X_1}(\rho(A''))$. De plus, d'après la définition de S_P, on a également $B_X(\rho(A)) = 1$ et $B_{X'}(\rho(A)) = 0$ et pour tout autre attribut A'', $B_{X'}(\rho(A'')) = B_X(\rho(A''))$. On a donc, pour tout i différent de $\rho(A)$ et $\rho(A')$, $B_X(i) = B_{X_1}(i) = B_{X'}(i)$ et

- $B_X(\rho(A)) = 1$ et $B_{X_1}(\rho(A)) = B_{X'}(\rho(A)) = 0$,
- $B_{X_1}(\rho(A')) = 1$ et $B_X(\rho(A')) = B_{X'}(\rho(A')) = 0$.

Par conséquent, $A \neq A'$, et donc soit $\rho(A) < \rho(A')$ soit $\rho(A') < \rho(A)$. Si $\rho(A) < \rho(A')$, par définition de S_P, on doit avoir $B_{X'}(\rho(A')) = B_X(\rho(A')) = 1$, ce qui contredit que $B_{X'}(\rho(A')) = 0$. De manière symétrique, on montre de même que $\rho(A') < \rho(A)$ est impossible, et donc la preuve est terminée. □

Soit une classe générique C donnée, la proposition suivante permet de générer les successeurs de C obtenus par application de la proposition 3.6(3) (Selection) sans redondance.

Proposition 4.2. *Soit $C = \langle X, Y, r\rangle$ une classe générique, et S_S l'ensemble des classes génériques défini par $S_S(C) = S_S^1(C) \cup S_S^2(C)$ tel que, pour tout $C' = \langle X', Y', r\rangle$ de S_S, on a $X' = X$ et $Y' = YA$ où A est un attribut de X qui satisfait l'une des deux conditions suivantes :*

1. *$C' \in S_S^1(C)$ si et seulement si il existe i tel que $B_X(\rho(K_i)) = 1$, $\rho(D_{i1}) \leq \rho(A) \leq \rho(D_{ip_i})$ (*i.e., *A est un attribut de dimension) et $\rho(A) = \min_k\{k \mid B_{Y'}(k) = 1\}$.*
2. *$C' \in S_S^2(C)$ si et seulement si pour tout i, $B_X(K_i) = 1$, $\rho(M_1) \leq \rho(A) \leq \rho(M_m)$ (*i.e., *A est une mesure) et $\rho(A) = \min_k\{k \mid B_{Y'}(k) = 1\}$.*

Alors $S_S(C) \subseteq succ(C)$ et de plus, pour toute classe générique $C_1 = \langle X_1, Y_1, r\rangle$ distincte de C, on a $S_S(C) \cap S_S(C_1) = \emptyset$.

Démonstration. Il est facile de voir que, sous les hypothèses de la proposition, YA fermé et si X est fermé alors $X \setminus A$ est non fermé. Donc, $S_S(C) \subseteq succ(C)$ est une conséquence de la proposition 3.6(3).

Soit $C_1 = \langle X_1, Y_1, r\rangle$ une classe générique distincte de C telle que $S_S(C) \cap succ(C_1) \neq \emptyset$, et soit $C' = \langle X', Y', r\rangle$ dans $S_S(C) \cap S_S(C_1)$.

On a alors $X = X_1 = X'$ (car tout C' de $S_S(C)$ est tel que $X' = X$) et donc d'après la proposition 3.6.3 il existe A' tel que $Y' = YA'$ (car $C' \in S_S(C_1)$). Par suite $B_{Y'}(\rho(A')) = 1$, $B_{Y_1}(\rho(A') = 0$ et pour tout autre attribut $A'' \neq A'$, $B_{Y'}(\rho(A'')) = B_Y(\rho(A''))$. De plus, d'après la définition de S_S, on a également $B_Y(\rho(A)) = 0$ et $B_{Y'}(\rho(A)) = 1$ et pour tout attribut $A'' \neq A$, $B_Y(\rho(A)) = B_{Y'}(\rho(A''))$. On a donc pour tout $i \neq \rho(A)$ et $\rho(A')$, $B_Y(i) = B_{Y_1}(i) = B_{Y'}(i)$ et

– $B_Y(\rho(A)) = 0$ et $B_{Y_1}(\rho(A)) = B_{Y'}(\rho(A)) = 1$,
– $B_{Y_1}(\rho(A')) = 0$ et $B_Y(\rho(A')) = B_{Y'}(\rho(A')) = 1$.

Par conséquent, $A \neq A'$, et donc $\rho(A') < \rho(A)$ soit $\rho(A) < \rho(A')$. Si $\rho(A') < \rho(A)$, par définition de S_S on doit avoir $B_{Y'}(\rho(A')) = B_Y(\rho(A')) = 0$ ce qui contredit $B_Y(\rho(A')) = 1$. De même de manière symétrique, on montre de même que $\rho(A) < \rho(A')$ est impossible, ce qui termine la preuve.

□

Soit une classe générique C donnée, la proposition suivante permet de générer des successeurs de C obtenus par application de la proposition 3.6(4) (Projection-Selection) sans redondance.

Proposition 4.3. *Soit $C = \langle X, Y, r\rangle$ une classe générique, et S_{PS} l'ensemble des classes génériques défini par $S_{PS}(C) = S^1_{PS}(C) \cup S^2_{PS}(C) \cup S^3_{PS}(C) \cup S^4_{PS}(C)$ tel que, pour tout $C' = \langle X', Y', r\rangle$ de S_{PS}, on a $X' = XA$ et $Y' = YA$ où A est un attribut de $U \setminus X$ satisfait l'une des quatre conditions qui satisfait les items 1,2,3 et on a $X' = R$ et $Y' = YA$ où A est un attribut de X qui satisfait l'item 4 :*

1. *$C' \in S^1_{PS}(C)$ si et seulement si il existe i tel que $\rho(D_{i1}) \leq \rho(A) \leq \rho(D_{ip_i})$ (*i.e., *A est un attribut de dimension), $B_X(\rho(A)) = 0$ et $\rho(A) = \min_k\{k \mid B_{Y'}(k) = 1\}$.*
2. *$C' \in S^2_{PS}(C)$ si et seulement si, $B_X(\rho(A)) = 0$, $\rho(M_1) \leq \rho(A) \leq \rho(M_m)$ (*i.e., *A est une mesure) et $\rho(A) = \min_k\{k \mid B_{Y'}(k) = 1\}$.*
3. *$C' \in S^3_{PS}(C)$ si et seulement si il existe A tel que $\rho(A) < n$ (*i.e., *A est une clé), $B_X(\rho(A)) = 0$, $K \setminus A \not\subseteq X$ et $\rho(A) = \min_k\{k \mid B_{Y'}(k) = 1\}$.*
4. *$C' \in S^4_{PS}(C)$ si et seulement si, $\rho(A) < n$ (*i.e., *A est une clé), $B_X(\rho(A)) = 0$, XA et YA fermé, $K \setminus A \subseteq X$, $M \cap (X \setminus Y) = \emptyset$ et $\rho(A) = \min_k\{k \mid B_{Y'}(k) = 1\}$.*

Alors $S_{PS}(C) \subseteq succ(C)$ et de plus, pour toute classe générique $C_1 = \langle X_1, Y_1, r\rangle$ distincte de C, on a $S_{PS}(C) \cap S_{PS}(C_1) = \emptyset$.

Démonstration. Il est facile de voir que, sous les hypothèses 1,2 et 3 de la proposition, puisque $A = \rho(A) = \min_k\{k \mid B_{Y'}(k) = 1\}$, si X est fermé alors $X' = XA$ est fermé et $Y' = YA$ est fermé. De même, sous l'hypothèse 4, puisque $\rho(A) = \min_k\{k \mid B_{Y'}(k) = 1\}$ alors $Y' = YA$ est fermé. Donc, $S_{PS}(C) \subseteq succ(C)$ est une conséquence de la proposition 3.6(4).

Soit $C_1 = \langle X_1, Y_1, r\rangle$ une classe générique distincte de C telle que $S_{PS}(C) \cap S_{PS}(C_1) \neq \emptyset$, et soit $C' = \langle X', Y', r\rangle$ dans $S_{PS}(C) \cap S_{PS}(C_1)$.

On a $C_1 = \langle X_1, Y_1, r\rangle$, $C = \langle X, Y, r\rangle$ et $C' = \langle X', Y', r\rangle$. En plus, $X' = X_1A'$, $Y' = Y_1A'$, $A' \notin X_1$, et $A' \notin Y'$, ainsi que $X' = XA$, $Y' = YA$, $A \notin X$, et $A \notin Y$. Puisque $\rho(A) = \rho_(A) = \min_k\{k \mid B_{Y'}(k) = 1\}$, on a alors $A = A'$. Nous avons $Y_1A' = Y' = YA$, $A' \notin Y'$, $A \notin Y$ et $A = A'$. Donc $Y = Y_1$ De même façon, on a $X_1 = X$. Ce qui est en contradiction avec le fait que $C \neq C_1$. Ce qui termine la preuve.

□

De plus, afin d'éviter les cas de redondances illustrés par les figures 4.6 et 4.7. Nous allons montrer grâce à la remarque 4.1 que l'application de la proposition 3.6(3) puis de la proposition 3.6(2) à une classe générique C, peut être remplacé par l'application de la proposition 3.6(2) puis de la proposition 3.6(3) sur C, ou (exclusif) par l'application de la proposition 3.6(2) puis de la proposition 3.6(2) puis de la proposition 3.6(4) sur C.

Remarque 4.1. *Soit $\langle X, Y, r\rangle$ une classe générique, par application de la proposition 3.6(3) nous obtenons des successeurs de la forme $\langle X, YA, r\rangle$ tels que $A \in X$, $X \setminus A$ non fermé et YA fermé, d'où : $(A \in D_i \setminus K_i$ et $K_i \in X)$ ou $(A \in M$ et $K \subseteq X)$. Puis en appliquant la proposition 3.6(2) à $\langle X, YA, r\rangle$, nous obtenons des successeurs de $\langle X, YA, r\rangle$ tels que $A' \in X$, $\langle X \setminus A', YA, r\rangle$ tels que $X \setminus A'$ fermé et $YA \subset X \setminus A'$, d'où :*

1. *si $(A \in D_i \setminus K_i$ et $K_i \in X)$ alors $A' \notin D_i \setminus K_i$, ainsi nous avons $A' \notin D_i$ ou $A' = K_i$ et*
2. *si $(A \in M$ et $K \subseteq X)$ alors il existe un j tel que $A' = K_j$.*

1. *Supposons tout d'abord que $(A \in D_i \setminus K_i$ et $K_i \in X)$. Dans ce cas, comme $X \setminus A'$ fermé et $Y \subseteq X \setminus A'$, nous pouvons appliquer la proposition 3.6(2) à la classe générique $\langle X, Y, r\rangle$, nous obtenons la classe générique $\langle X \setminus A', Y, r\rangle$ ce qui implique que :*
   - *si $A' \notin D_i$ alors $(X \setminus A') \setminus A$ est non fermé, de plus comme YA fermé, nous pouvons appliquer la proposition 3.6(3), nous obtenons alors la classe générique $\langle X \setminus A', YA, r\rangle$ qui est la même classe générique que celle obtenue par application de proposition 3.6(3) puis proposition 3.6(2).*
   - *si $(A' = K_i)$ en appliquant la proposition 3.6(2) à la classe générique $\langle X, Y, r\rangle$ puisque $X \setminus K_i$ est fermé, nous obtenons la classe générique $\langle X \setminus K_i, Y, r\rangle$, puis comme $\langle X \setminus K_i\rangle \setminus A$ est fermé. Nous pouvons appliquer la proposition 3.6(2), nous obtenons alors la classe générique $\langle (X \setminus K_i) \setminus A, Y, r\rangle$. Finalement, puisque YA fermé, nous pouvons appliquer la proposition 3.6(4), ce qui permet d'obtenir la classe générique $\langle X \setminus K_i, YA, r\rangle$ qui est la même classe générique que celle obtenue par application de proposition 3.6(3) puis proposition 3.6(2).*
2. *Supposons maintenant que $(A \in M$ et $K \subseteq X)$ dans ce cas, comme nous l'avons vu précédemment, il existe j tel que $A' = K_j$. Comme $X \setminus K_j$ est fermé, nous pouvons appliquer la proposition 3.6(2) à la classe générique $\langle X, Y, r\rangle$, nous obtenons la classe générique $\langle X \setminus K_j, Y, r\rangle$. De plus, comme $A \in M$ et $K_j \notin X \setminus K_j$ alors $(X \setminus K_j) \setminus A$ est fermé. Nous pouvons donc appliquer la proposition 3.6(2), et nous obtenons la classe générique $\langle (X \setminus K_j) \setminus A, Y, r\rangle$. Finalement, puisque $X \setminus K_j$ et YA sont fermés, nous pouvons appliquer la proposition 3.6(4), nous obtenons ainsi la classe générique $\langle X \setminus K_j, YA, r\rangle$ qui est la même classe générique que celle obtenue par application de proposition 3.6(3) puis proposition 3.6(2).*

Ce qui termine la preuve.

Les propostions 4.4, 4.5, 4.6 de la section suivante montre que l'implémentation de notre algorithme `FQF`, qui est basée sur les propositions 4.1, 4.2 et 4.3 et la remarque 4.1, génère l'ensemble des classes génériques.

4.5.4 Complétude de l'implémentation de notre algorithme FQF

Dans toutes les propositions r représente la table considérée (une table de dimension ou la table de jointure) et U l'ensemble des attributs de la table r.

Proposition 4.4. *X est un schéma fermé, $m = |U| - |X|$. $\forall m$, $1 \leq m < |U|$, la classe $\langle X, \emptyset, r\rangle$ peut être engendré par l'implémentation de proposition 4.1.*

Démonstration. Soit $E = \{k | \forall l, k < l \leq |U|, B_X(l) = 1\}$. $\rho(A) = minE$ si $E \neq \emptyset$ sinon $\rho(A) = |U|$. La valeur $minP_X$ est le minimum de E_X, si $E_X \neq \emptyset$. Sinon, $minP_X = |U|$. Donc, $minP_X$ est la position du plus grand $\rho(A)$, $A \in U$ tel que $B_X(\rho(A)) = 0$, si $X \neq U$.

Pour le cas $m = 1$, puisque X est fermé, $\exists K_i \in U, X = U \setminus K_i$. Donc $\langle X, \emptyset, r\rangle \in S^1_P(\langle U, \emptyset, r\rangle)$ et $\rho(K_i) = minP_X$.
Supposons que le cas $m = k < |U| - 1$ est vrai.
Pour le cas $m = k + 1$, on traite seulement $minP_X > 1$. Car $minP_X = 1$ est traité dans le cas $m = 1$.
Soit $X' = XA$ où $\rho(A) = minP_X$. Donc $B_X(k) = B_{X'}(k)$ si $k \neq minP_X$ et $B_{X'}(minP_X) = 1$. Ici, on ne traite que le cas $X' \neq U$, car $X' = U$ est déjà traité dans le cas $m = 1$.
Nous montrons d'abord que X' est fermé. Si X' n'est pas fermé, il y a deux cas :

1. $\exists D_{i,j} \notin X'$, mais $K_i \in X'$ ou
2. $\forall K_i, K_i \in X'$, mais $\exists M_j \notin X'$.

Dans tous les deux cas, X n'est pas fermé car $\rho(A) = minP_X \geq D_{ij}$ ou $\rho(A) = minP_X \geq M_j$. Donc, il y a une contradiction.

Par conséquent, la classe générique $\langle X', \emptyset, r\rangle$ peut être engendrée par l'hypothèse $m = k$. Car $|X'| + k = |U|$. Donc la classe générique $\langle X, \emptyset, r\rangle = \langle X' \setminus A, \emptyset, r\rangle$ peut être engendrée grâce à l'implémentation de la proposition 4.1. En plus, si $A = K_i$, alors $\langle X, \emptyset, r\rangle \in S^1_P(\langle X', \emptyset, r\rangle)$. Si $A = D_{i,j}$, alors $\langle X, \emptyset, r\rangle \in S^2_P(\langle X', \emptyset\rangle)$ et si $A = M_i$, alors $\langle X, \emptyset, r\rangle \in S^3_P(\langle X', \emptyset, r\rangle)$.

Ce qui termine la preuve. □

Proposition 4.5. *La classe générique $\langle X, A, r\rangle$ peut être engendrée par l'implémentation de proposition 4.2 ou de la proposition 4.3 à partir d'une classe générique engendrée par l'implémentation de proposition 4.1.*

Démonstration. Si $\langle X, A, r\rangle$ est une classe générique, alors X est fermé.

1. $A = Mi$.
   Si $X = U$, alors $\langle X, A, r\rangle \in S^2_S(\langle X = U, \emptyset, r\rangle)$. Si $X \neq U$, alors $X \setminus A$ est fermé. car il existe au moins un $K_i \notin X$. Donc, $\langle X \setminus A, \emptyset, r\rangle\rangle$ est une classe générique engendrée par l'implémentaion de proposition 4.1 et $\langle X, A, r\rangle \in S^2_{PS}(\langle X \setminus A, \emptyset, r\rangle)$.
2. $A = D_{ij}$.
   Si $K_i \in X$, alors $\langle X, D_{i,j}, r\rangle \in S^1_S(\langle X, \emptyset, r\rangle)$. Sinon, $X \setminus D_{i,j}$ est fermé et $\langle X \setminus D_{i,j}, \emptyset, r\rangle$

est une classe générique engendrée par l'implémentaion de proposition 4.1. Donc, $\langle X, D_{i,j}, r\rangle \in S^1_{PS}(\langle X \setminus D_{i,j}, \emptyset, r\rangle)$.

3. $A = K_i$.
   Nous avons donc que $D_i = \{K_i\}$. Si $X \neq U$, alors $X \setminus K_i$ est fermé et $\langle X \setminus K_i, \emptyset, r\rangle$ est une classe générique engendrée par l'implémentaion de la Proposition 4.1. Par conséquent, $\langle X, K_i, r\rangle \in S^3_{PS}(\langle X \setminus K_i, \emptyset, r\rangle)$. Sinon, $(U \setminus K_i) \setminus \{M_1, ..., M_m\}$ est fermé $\langle (U \setminus K_i) \setminus \{M_1, ..., M_m\}, \emptyset, r\rangle$ est une classe générique engendrée par l'implémentaion de proposition 4.1. Donc, $\langle U, K_i, r\rangle \in S^4_{PS}(\langle (U \setminus K_i) \setminus \{M_1, ..., M_m\}, \emptyset, r\rangle)$.

□

Proposition 4.6. *La classe générique $\langle X, Y, r\rangle$ où $1 < |Y| \leq |U|$ peut être engendrée par l'implémentation de proposition 4.2 ou la proposition 4.3 à partir d'une classe générique engendrée par l'implémentation de proposition 4.2 ou la proposition 4.3.*

Démonstration. Soit $Y = AY'$, $A \notin Y'$ et $B_Y(\rho(A)) = 1$ et $\forall k < \rho(A), B_Y(k)) = 0$. Soit $Y' = BY''$, $A \neq B, B \notin Y''$ et $B_{Y'}(\rho(B)) = 1$ et $\forall k < \rho(B), B_{Y'}(k)) = 0$. On a $\rho(A) < \rho(B)$
Le cas $|Y| = 2$

1. $A = M_i$.
   Puisque $A = M_i$, $B = M_j$ et $\{B\}$ est fermé. Si $X = U$, alors $\langle X = U, B, r\rangle$ est une classe engendrée par l'implémentation de proposition 4.2 ou 4.3. En plus, $\langle X, AB, r\rangle \in S^2_S(\langle X = U, B, r\rangle)$. Si $X \neq U$, alors $(X \setminus A)$ est fermé et $\langle X \setminus A, B, r\rangle$ est une classe engendrée par l'implémentation de proposition 4.2 ou 4.3. Donc, $\langle X, AB, r\rangle \in S^2_{PS}(\langle X \setminus A, B, r\rangle)$.
2. $A = D_{ij}$.
   $\{B\}$ est fermé car $B = D_{i',j'}$ ou $B = M_j$. Si $K_i \in X$, alors $\langle X, D_{i,j}B, r\rangle \in S^1_S(\langle X, B, r\rangle)$. Sinon, $X \setminus D_{i,j}$ est fermé et $\langle X \setminus D_{i,j}, B, r\rangle$ est une classe générique engendrée par l'implémentaion de la proposition 4.2 ou de la proposition 4.3. Donc, $\langle X, D_{i,j}B, r\rangle \in S^1_{PS}(\langle X \setminus D_{i,j}, B, r\rangle)$.
3. $A = K_i$. $\{B\}$ est fermé. Car si $B = K_j$ alors $D_j = \{K_j\}$ et les autres situations sont discutées dans le cas $A = D_{ij}$. Si $X \neq U$ supposons que $B \neq M_i$, alors $X \setminus K_i$ est fermé et $\langle X \setminus K_i, B, r\rangle$ est une classe générique engendrée par l'implémentation de proposition 4.2 ou de la proposition 4.3. Par conséquent, $\langle X, K_iB, r\rangle \in S^3_{PS}(\langle X \setminus K_i, B, r\rangle)$. Sinon, supposons que $B \neq M_i$, $(U \setminus K_i) \setminus \{M_1, ..., M_m\}$ est fermé $\langle (U \setminus K_i) \setminus \{M_1, ..., M_m\}, B, r\rangle$ est une classe générique engendrée par l'implémentaion de proposition 4.2 ou de la proposition 4.3. Donc, $\langle U, K_iB, r\rangle \in S^4_{PS}(\langle (U \setminus K_i) \setminus \{M_1, ..., M_m\}, B, r\rangle)$. Dans le cas $B = M_i$, $(U \setminus K_i) \setminus (\{M_1, ..., M_m\} \setminus B)$ est fermé et $\langle (U \setminus K_i) \setminus (\{M_1, ..., M_m\} \setminus B), B, r\rangle$ est une classe générique engendrée par l'implémentaion de la proposition 4.2 ou de la proposition 4.3. $\langle U, K_iB, r\rangle \in S^4_{PS}(\langle (U \setminus K_i) \setminus (\{M_1, ..., M_m\} \setminus B), B, r\rangle)$.

Supposons que les cas $2 \leq |Y| = k$ sont vrais.
Dans le cas $|Y| = k + 1 \leq |U|$ $A' = D_{i'j'}$ ou $A' = M_j$. Soit $Y = ABY''$

1. $A = M_i$.
   Puisque $A = M_i$, $Y \subseteq \{M_1, ..., M_m\}$. Donc BY'' est fermé. Si $X = U$, alors

$\langle X = U, BY'', r\rangle$ est une classe engendrée par l'hypothèse des cas $|Y| = k$. En plus, $\langle X, Y, r\rangle \in S_S^2(\langle X = U, BY'', r\rangle)$. Si $X \neq U$, alors $(X \backslash A)$ est fermé et $\langle X \backslash A, BY'', r\rangle$ est une classe engendrée par l'hypothèse. Donc, $\langle X, Y, r\rangle \in S_{PS}^2(\langle X \setminus A, BY', r'\rangle)$.

2. $A = D_{ij}$.
   BY'' est fermé car $\forall A' \in BY'', A' = D_{i'j'}$ ou $A' = M_j$". Si $K_i \in X$, alors $\langle X, Y, r\rangle \in S_S^2(\langle X, BY'', r\rangle)$. Sinon, $X \backslash D_{ij}$ est fermé et $\langle X \backslash D_{ij}, BY'', r\rangle$ est une classe générique engendrée par l'hypothèse. Donc, $\langle X, Y, r\rangle \in S_{PS}^2(\langle X \setminus D_{i,j}, BY'', r\rangle)$.

3. $A = K_i$.
   BY'' est fermé de la même manière que dans le cas $|Y| = 2$
   Si $X \neq U$, alors $X \setminus K_i$ est fermé et $\langle X \setminus K_i, BY'', r\rangle$ est une classe générique engendrée par l'hypothèse. Par conséquent, $\langle X, Y, r\rangle \in S_{PS}^3(\langle X \setminus K_i, BY'', r\rangle)$. Si $X = U$, alors $(U \setminus K_i) \setminus (\{M_1, ..., M_m\} \setminus BY'')$ est fermé $\langle (U \setminus K_i) \setminus (\{M_1, ..., M_m\} \setminus BY''), BY'', r\rangle$ est une classe générique engendrée par l'hypothèse ainsi $\langle U, Y, r\rangle \in S_{PS}^4(\langle (U \setminus K_i) \setminus (\{M_1, ..., M_m\} \setminus BY''), BY'', r\rangle)$.

Ce qui termine la preuve. □

Dans cette section, nous avons vu, grâce aux propositions 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 et à la remarque 4.1 , que nous pouvons générer l'ensemble des successeurs en évitant partiellement certaines redondances. Notre implémentation utilise ces résultats pour optimiser notre algorithme. Cependant, nous n'avons pas pu prouver l'inexistence d'autres redondances. Une étude ultérieure des éventuelles redondances que nous pourrions avoir est néanmoins envisagée.

4.6 Tests effectués

Dans cette section, nous présentons les résultats des différents tests que nous avons effectués sur des bases de données synthétiques et réelles. Nous avons effectué nos expériences sur un ordinateur PC Pentium Duo Core ayant 4 Go de mémoire vive et tournant sur le système d'exploitation Ubuntu Linux 2.6. Il faut noter que nous n'allouons à la Java Virtual Machine que 2 Go. Comme mentionné précédemment, l'algorithme est implémenté en Java et la communication avec la base de donnée MySQL se fait par jdbc.
Nos bases de données de test sont générées en utilisant notre propre générateur qui est une adaptation du générateur d'IBM dans le cas d'une base de données ayant plusieurs tables et définie sur un schéma étoile. Ce générateur est disponible à l'adresse(`http://www.almaden.ibm.com`).
Les bases de données ayant un schéma étoile générées sont notées db`d`Da`t`Tt`m`Mm avec `d` représentant le nombre de dimensions, `a` représentant le nombre total d'attributs, `t` représentant le nombre de tuples dans la table de faits, et `m` représentant le nombre d'attributs de mesure.
Dans nos expériences, si nous ne le précisons pas, le support est fixé à 0.6 multiplié par le nombre de tuples de la table de faits, c'est à dire $0.6 \times$ `t`.
Nous noterons également, que si nous ne le précisons pas, les temps d'exécution de nos expériences incluent le temps mis pour la construction de la table auxiliaire AUX.

4.6.1 Performances de FQF

La figure 4.8 compare le temps d'exécution de `FQF` et notre implémentation précédente `DBQueriesIDEAS` ([66]) pour db2D12TtM1, avec `t` compris 50 et 5 000. Sur cette figure, nous remarquons clairement que `FQF` est plus efficace, le temps d'exécution est réduit d'au moins de 33%. Nous pouvons noter également que si l'on exclut le temps mis pour le calcul de AUX, le temps d'exécution est toujours inférieur à 40 secondes quel que soit le nombre de tuples de la table de faits. De plus, il faut noter également que `DBQueriesIDEAS` envoie une erreur mémoire (`OutOfMemory Exception`) dès que le nombre de tuples de la table de faits dépasse 5 000, contrairement à `FQF`. Ceci est dû au fait que `DBQueriesIDEAS` maintient une liste pour chaque classe générique et la taille de ces listes devient très grande au cours de l'exécution de l'algorithme.

La figure 4.9 montre le temps mis par l'algorithme `FQF` pour rechercher les classes fréquentes et le temps mis pour le calcul de la table AUX, pour les bases de données db2D12TtM1 avec `t` compris entre 10 000 et 90 000. Nous pouvons remarquer que le temps mis pour la découverte des requêtes fréquentes est faible lorsque la table AUX est connue. De plus, nous remarquons que lorsque la table AUX est connue, le temps de recherche des requêtes fréquentes augmente lentement lorsque la taille de la table de jointure augmente.

La figure 4.10 montre le temps d'exécution de notre algorithme en fonction du nombre de dimensions, pour les bases de données db`d`D12T2000M1 pour `d` variant entre 2 et 5 et pour un seuil de support de 6 000. Cette figure montre clairement que le temps mis pour la recherche des classes de requêtes fréquentes diminue lorsque le nombre de dimensions augmente. Ceci est dû au fait que, pour un nombre d'attributs fixé (12 dans notre cas), lorsque le nombre de dimensions `d` augmente, plus de dépendances fonctionnelles sont disponibles et ainsi, moins de classes candidates sont traitées.
Il est important de noter que conformément à notre affirmation précédente le nombre de passes sur la base est en $O(N \times |U|)$ (avec N représentant le nombre de dimensions et $|U|$ représentant nombre total d'attributs), ce qui fait qu'on devait s'attendre à ce que le temps d'exécution augmente lorsque N augmente.
Cependant, cette expérience montre que, lorsque le nombre de dimensions augmente le nombre de passes augmente également, mais cette augmentation du nombre de passes est compensée par la réduction drastique du nombre de classes génériques candidates due à l'augmentation du nombre de dépendances fonctionnelles. En effet, lorsque le nombre de dépendances fonctionnelles augmente, le nombre de comparaison possibles augmente également, d'où le nombre d'élagage.

La figure 4.11 montre le temps d'exécution de notre algorithme en fonction du nombre de mesures. Comme le montre cette figure, le temps d'exécution augmente très rapidement lorsque le nombre de mesures augmente. En effet, lorsque le nombre de mesures augmente, le nombre de classes génériques candidates augmente de manière drastique.

Les figures 4.12 et 4.13 donnent les temps d'exécution de FQF en utilisant trois méthodes :

– $AUX(J)$ est calculé pour chaque instance de base de données ;
– $AUX(J)$ est calculé de manière incrémentale quand nous passons d'une base de données à la suivante ayant plus de tuples ;
– en considérant que la table $AUX(J)$ est un pré-calcul et dans ce cas, le temps mis pour le calcul de $AUX(J)$ n'est pas pris en compte.

Dans les deux cas, nous considérons un schéma étoile ayant 12 attributs et une mesure, les bases de données sont alors de la forme db2D12TtM1. Sur la figure 4.12 `t` varie de 500 à 5 000 et le support minimum est égal à 100, et dans la figure 4.13, `t` varie de 10 000 à 100 000 et le support minimum est égal à 1 000. Ces deux figures montrent que le calcul incrémental est nettement plus efficace que le calcul non incrémental.

La figure 4.14 montre le temps d'exécution de `FQF` en fonction du seuil de support (exprimé en pourcentage par rapport à la taille de la table de faits), pour la base de données db5D25T10000M1 avec `t` égal à 2 000, 5 000 et 10 000.
Nous remarquons que le temps d'exécution diminue rapidement lorsque le support augmente, ceci est dû au fait que le nombre de classes candidates est réduit.

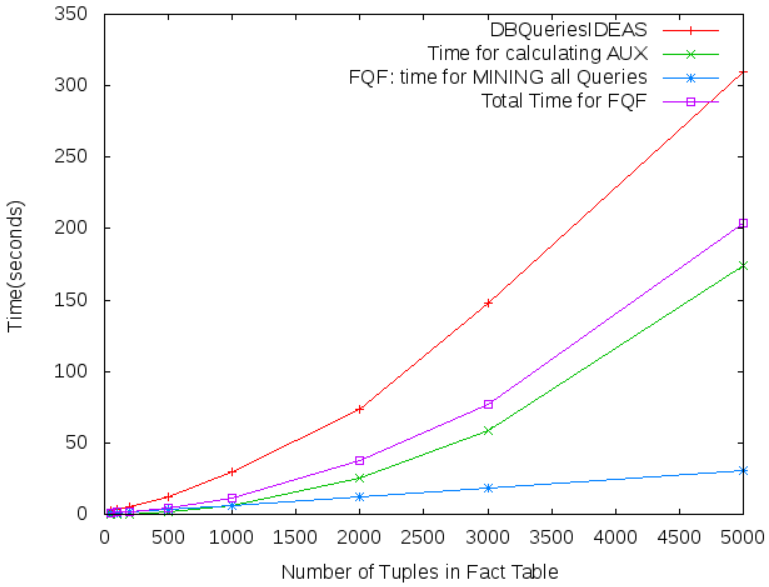


FIGURE 4.8 – Comparaison de notre implémentation à celle précédente

Dans la section suivante, nous allons donner quelques résultats comparatifs de notre algorithme relativement à `Conqueror` et `Conqueror`$^+$.

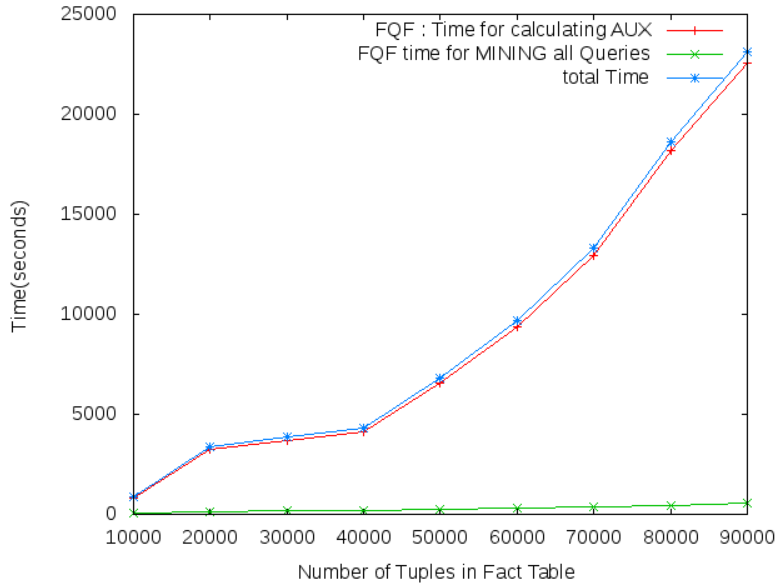


FIGURE 4.9 – Temps d'exécution de FQF en fonction du nombre de tuples dans la table de faits.

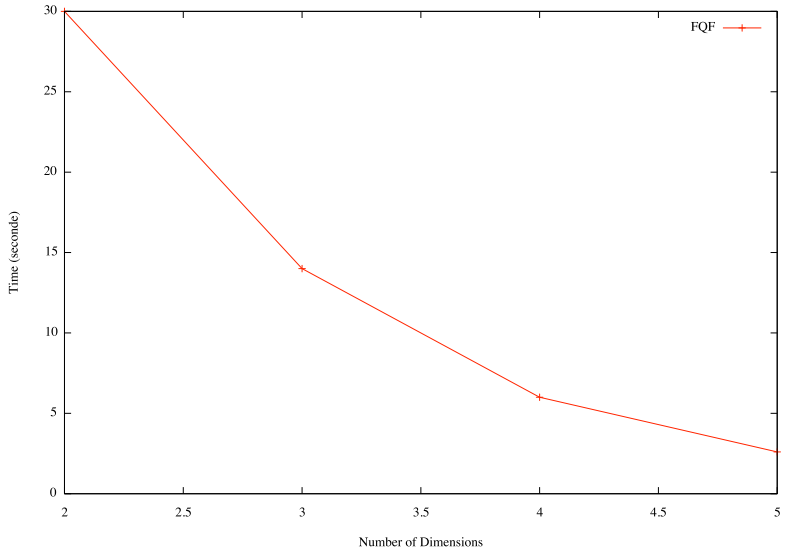


FIGURE 4.10 – Temps d'exécution de FQF en fonction du nombre de dimensions.

4.6.2 Résultats comparatifs de `FQF`, `Conqueror` et `Conqueror`$^+$

Nous présentons ici, les résultats comparatifs de `FQF`, de `Conqueror` et de `Conqueror`$^+$ sur différentes instances de bases de données synthétiques de type schéma étoile, en prenant soin toutefois, de ne considérer que les égalités entre clés primaires et clés étrangères au niveau de la jointure pour l'ensemble des trois implémentations.

Globalement, nous remarquons que, pour l'ensemble des deux figures (4.15, 4.16), le temps d'exécution de notre algorithme est toujours inférieur aux temps d'exécution de

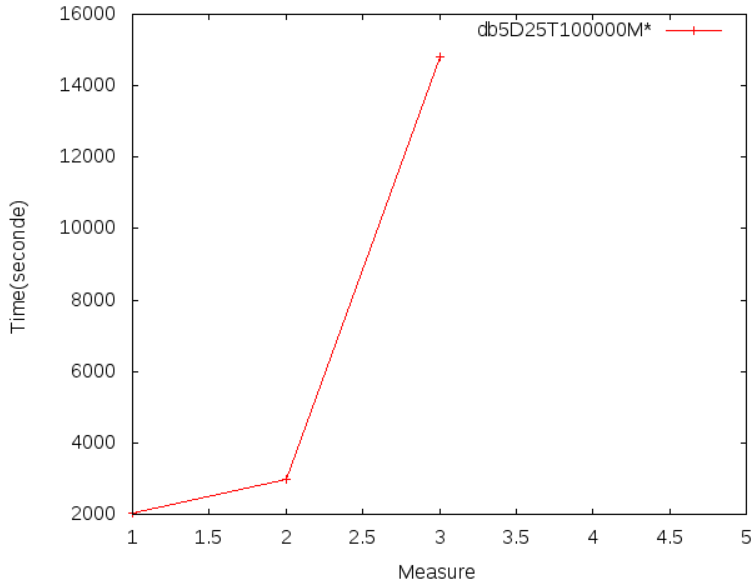


FIGURE 4.11 – Temps d'exécution de FQF en fonction du nombre de mesures.

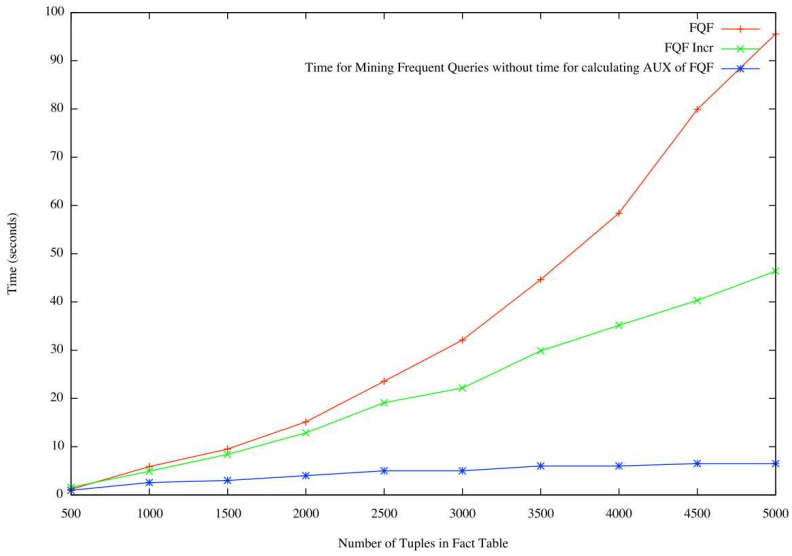


FIGURE 4.12 – Temps d'exécution de FQF en fonction du nombre de tuples de la table de faits pour de petites bases.

`Conqueror`$^+$, qui eux mêmes sont inférieurs aux temps d'exécution de `Conqueror`.Les différences sont dues au fait que `Conqueror` ne prend pas en compte les dépendances fonctionnelles. Ainsi, des requêtes équivalentes relativement au dépendances fonctionnelles et au dépendances d'inclusion sont évaluées séparément alors qu'une seule évaluation serait suffisante, puisque deux requêtes équivalentes ont même support .

D'autre part, `Conqueror` et `Conqueror`$^+$ ont trois boucles séparées, une boucle de Projection, une boucle de Jointure et une boucle de Sélection. Ceci implique que l'élagage

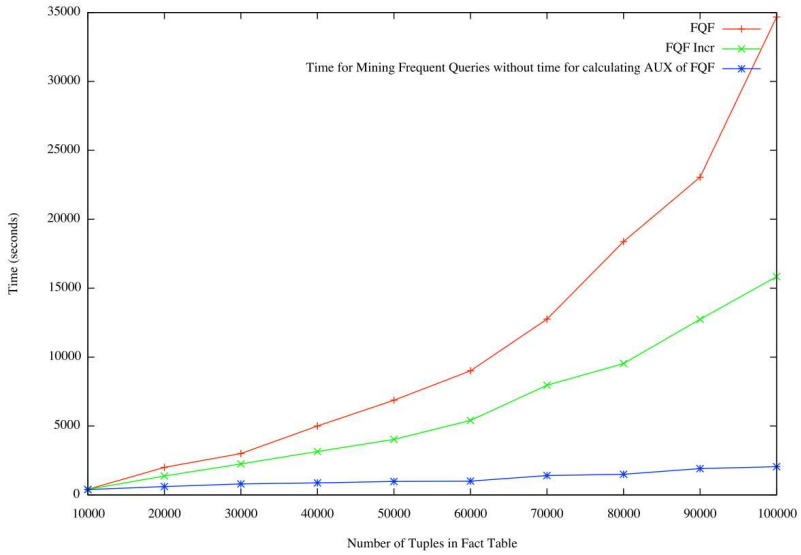


FIGURE 4.13 – Temps d'exécution de FQF en fonction du nombre de tuples de la table de faits pour de grandes bases.

au niveau de ces deux algorithmes est moins efficace que le nôtre. En effet, les requêtes ne sont pas générées selon l'ordre défini sur les classes d'équivalence.

De plus, l'évaluation des requêtes candidates par `Conqueror`$^+$ d'un niveau donné se fait grâce à plusieurs requêtes SQL, ce qui implique plusieurs passes sur la base de données malgré quelques optimisations proposées par les auteurs, contrairement à notre algorithme qui effectue une seule passe, grâce à l'utilisation d'une table auxiliaire.

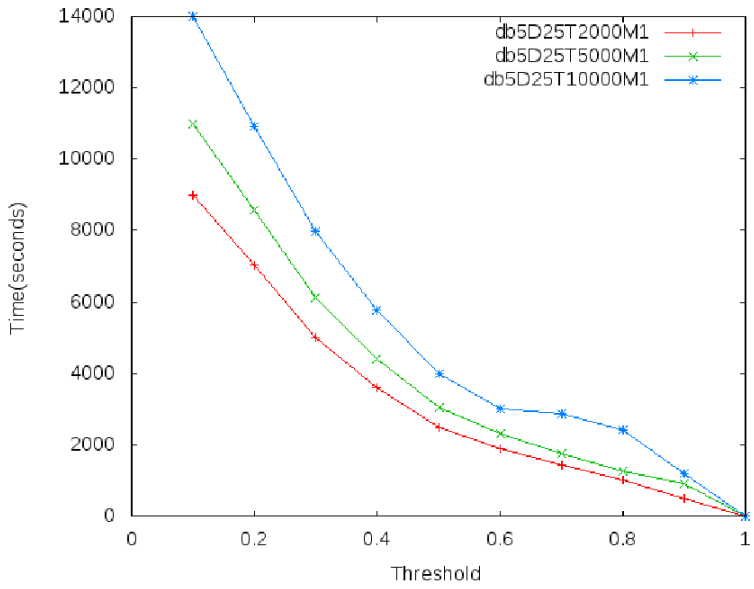


FIGURE 4.14 – Temps d'exécution en fonction du seuil de support.

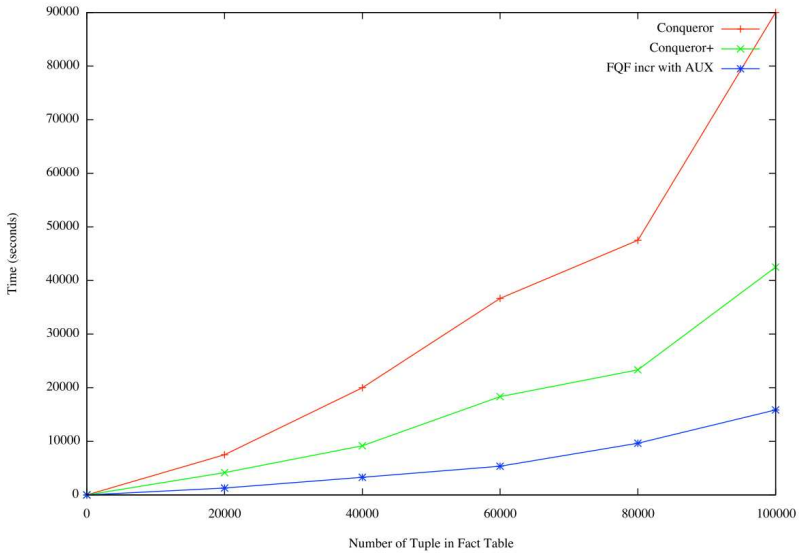


FIGURE 4.15 – Comparaison du temps d'exécution de `FQF`, `Conqueror`$^+$ et `Conqueror` en fonction du seuil de support

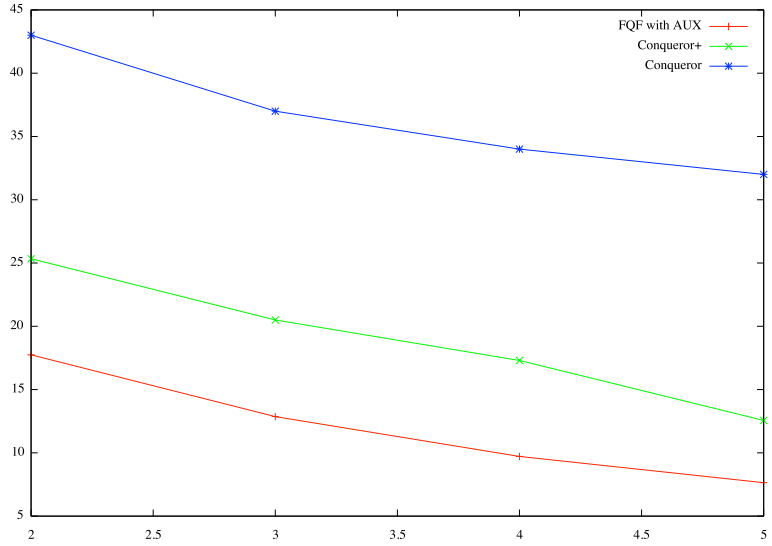


FIGURE 4.16 – Comparaison du temps d'exécution de `FQF`, `Conqueror`$^+$ et `Conquerors` en fonction du nombre de dimensions

Nous avons également effectué des tests sur des données réelles en utilisant un extrait de la base de données `imdb`.

La figure 4.17 montre que notre algorithme est plus efficace que `Conqueror` . Comme il a été dit précédemment, ceci est dû au fait que`Conqueror` ne prend pas en compte les dépendances foncttionnelles et les dépendances d'inclusion. Cependant nous rappelons, que cette méthode considère aussi les requêtes dont la condition de sélection est de la forme $Y' = Y$.

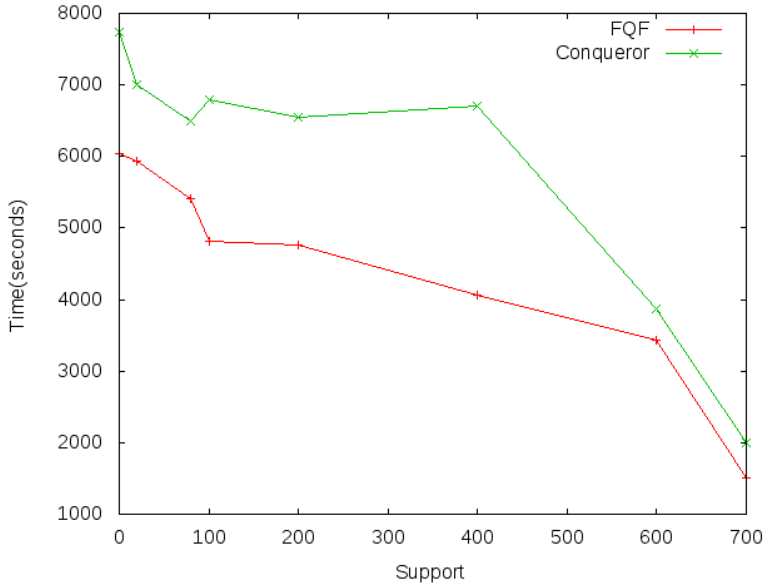


FIGURE 4.17 – Comparaison du temps d'exécution de FQF et Conqueror en fonction du seuil de support

4.7 Conclusion

Nous avons présenté dans cette partie, l'implémentation de notre algorithme FQF ainsi que différents tests sur des données synthétiques et réelles. Notre algorithme a été également comparé à des algorithmes de recherche de requêtes fréquentes existants.
Nous avons pu remarquer que notre algorithme est efficace et permet de trouver en un temps raisonnable l'ensemble des requêtes fréquentes surtout lorsque la table AUX est connue.
Nous remarquons également que notre implémentation est passe à l'échelle lorsque le nombre d'attributs ou la taille de la table de faits augmente.

5
Conclusions et Perspectives

Le problème de la recherche de motifs intéressants dans les bases de données est un problème difficile. De plus, la plupart des méthodes existantes sont dédiées au problème de la recherche d'itemsets fréquents dans les bases de données transactionnelles ou au problème de la recherche de requêtes fréquentes dans le cas où l'objet de comptage encore appelé schéma de projection est fixe. Malheureusement, ces techniques ne permettent pas de trouver certains types de motifs comme les règles d'association mettant en jeu des objets de comptage différents, les dépendances fonctionnelles, les dépendances fonctionnelles approximatives, les dépendances fonctionnelles conditionnelles et les dépendances fonctionnelles conditionnelles approximatives. Cependant, ces motifs peuvent être obtenus grâce à des requêtes fréquentes définies sur des schémas de projection différents. C'est ainsi que cette thèse s'intéresse au problème de la recherche de requêtes ayant des schémas de projection variables dans les bases de données définies sur un schéma étoile.

Dans les sections suivantes, nous allons tout d'abord résumer nos contributions puis nous allons donner quelques directions de recherches pouvant être envisagées.

5.1 Résumé de nos contributions

Dans cette thèse, nous nous sommes intéressés au problème de la recherche de requêtes fréquentes dans les bases de données définies sur un schéma étoile. Nous avons ainsi proposé une méthode baptisée `FQF` qui permet de trouver l'ensemble des requêtes fréquentes de type Projection-Selection-Jointure (PSJ), où la condition de jointure est l'égalité entre une clé primaire et une clé étrangère, d'une base de données relationnelle définie sur un schéma étoile. Cette méthode utilise les dépendances fonctionnelles et les dépendances d'inclusion présentes dans les bases de données relationnelles. Dans la partie 2 du chapitre 3 nous avons proposé un formalisme pour la recherche des requêtes fréquentes de type PSJ, où la jointure est l'égalité entre une clé primaire et une clé étrangère, dans les bases de données relationnelles. Ensuite, dans la partie 3 du chapitre 3, nous avons proposé une relation de pré-ordre sur les requêtes basée sur les dépendances fonctionnelles et les dépendances d'inclusion. Cette relation de pré-ordre est anti-monotone par rapport à la mesure de support, ce qui permet d'utiliser un algorithme par niveau de type Apriori ([8]). De plus, cette relation de pré-ordre induit des classes d'équivalence sur l'ensemble des requêtes et grâce à l'anti-monotonie du support, nous avons montré que deux requêtes appartenant à la même classe d'équivalence ont même support. Nous avons alors proposé

une méthode pour générer et évaluer un représentant par classe d'équivalence, ce qui permet ainsi de générer et évaluer moins de requêtes candidates. Nous avons également proposé une méthode efficace basée sur une table auxiliaire AUX qui permet d'évaluer efficacement le support des classes de requêtes candidates sans pour autant utiliser des requêtes SQL qui sont peu efficaces en raison des délais de communications avec la base de données. Nous avons montré également que la construction de AUX peut être vue comme un pré-calcul.

Ensuite, nous avons proposé un algorithme par niveau appelé `FQF` capable de générer et d'élaguer efficacement l'espace de recherche dont le nombre de passes est linéaire par rapport à la la taille de l'univers de la base de données. Dans la partie implémentation, nous avons proposé une méthode pour gérer efficacement la mémoire centrale et générer les requêtes candidates en évitant certaines redondances.

Finalement, nous avons présenté quelques résultats expérimentaux pour prouver sa faisabilité. Des résultats comparatifs avec les deux méthodes similaires à notre approche ont été également présentées

5.2 Perspectives

Les travaux que nous avons menés durant cette thèse ouvrent la voie à plusieurs perspectives de recherches. Nous allons présenter dans cette section quelques unes d'entres elles :

Optimisation de notre implémentation. Notre implémentation pourrait être améliorée. En effet, il est possible d'effectuer la recherche dans les tables de dimension et la table de jointure simultanément. Une parallélisation de notre algorithme est ainsi envisagée pour de meilleurs temps d'exécution. De plus, une étude plus poussée des cas de redondances afin de les éliminer entièrement pourraait améliorer sensiblement les performances de notre algorithme.

Généralisation à d'autres types de bases de données Comme notre approche ne s'applique qu'aux bases de données définies sur un schéma étoile, nous envisageons de la généraliser aux bases de données définies su en flocons ou définies sur un schéma en constellation. Dans ce cadre, des travaux sont actuellement en cours.

Généralisation aux requêtes autres que les requêtes de type PSJ. Nous envisageons de considérer des requêtes plus générales que les requêtes conjonctives dont la condition de jointure est une égalité entre clé-primaire et clé étrangère dans ce cadre, il serait intéresssant d'étendre notre approche aux requêtes dont la condition de jointure est un ensemble d'égalités entre n'importe quel couple d'attributs de la base. Nous souhaitons également voir comment notre approche pourrait s'appliquer aux requêtes disjonctives. Nous souhaitons également intégrer des opérations complexes de types OLAP (Rollup, Rotate, Pull, Split, Drop...) aux requêtes considérées.

Optimisation de la recherche de requêtes fréquentes en cas de mise à jour de la base de données. Dans la version actuelle de notre implémentation, nous considérons les bases de données comme étant statiques, ce qui n'est pas le cas dans la réalité. Une perspective consiste à prendre en compte les mises à jour de la base de données pour optimiser la recherche des requêtes fréquentes de la nouvelle instance.

Visualisations des requêtes fréquentes découvertes. A cause du nombre important de requêtes fréquentes découvertes, la visualisation de celles-ci est également à envisager. Cette visualisation, nous permettra de représenter les requêtes fréquentes découvertes pour une fouille visuelle.
Intégration de notre approche aux SGBD existant. Il serait intéressant d'intégrer notre approche au noyau des SGBD existants. Ceci nous permettrait de supprimer les communications JDBC entre notre application et la base de données. Grâce à la suppression de cette communication, nous pourrions améliorer sensiblement le temps de calcul, car les entrées sorties avec la base de données représentent 70% du temps de calcul total.
Autres applications des requêtes fréquentes. Un autre point que nous souhaitons étudier ultérieurement consiste à voir s'il est possible d'obtenir d'autres types de motifs à partir des requêtes fréquentes découvertes hormis les règles d'association, les dépendances fonctionnelles normales ou approximatives et les dépendances fonctionnelles conditionnelles ou approximatives, que nous pourrions obtenir à partir des requêtes fréquentes découvertes. Une autre perspective consisterait à sauvegarder les requêtes extraites à des fins d'analyse en ligne par exemple, un utilisateur pourrait demander par exemple : les dépendances fonctionnelles ou les clés d'une base de données d'extraction données.

Bibliographie

[1] *Proceedings of the Fourteenth International Conference on Data Engineering, February 23-27, 1998, Orlando, Florida, USA*. IEEE Computer Society, 1998.

[2] Constraint-based rule mining in large, dense databases. In *Proceedings of the 15th International Conference on Data Engineering*, ICDE '99, pages 188–, Washington, DC, USA, 1999. IEEE Computer Society.

[3] *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, April 15-20, 2007, The Marmara Hotel, Istanbul, Turkey*. IEEE, 2007.

[4] *Proceedings of the SIAM International Conference on Data Mining, SDM 2008, April 24-26, 2008, Atlanta, Georgia, USA*. SIAM, 2008.

[5] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[6] R. Agrawal, T. Imieliński, and A. Swami. Mining association rules between sets of items in large databases. In *SIGMOD '93 : Proceedings of the 1993 ACM SIGMOD international conference on Management of data*, pages 207–216, New York, NY, USA, 1993. ACM.

[7] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[8] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In M. Jarke J. B. Bocca and C. Zaniolo, editors, *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, pages 487–499. Morgan Kaufmann, 12–15 1994.

[9] R. Agrawal and R. Srikant. Mining sequential patterns. In Yu and Chen [120], pages 3–14.

[10] A.J.Knobbe, H. Blockeel, A.P.J.M. Siebes, and D.M.G. Van der Wallen. Multi-relational data mining. Technical Report INS-R9908, ISSN, IBM Almaden Research Center, San Jose, CA, 1999.

[11] W. Armstrong. Dependency structures of data base relationships. In IFIP Congress*, pages 580–583. North Holland*, 1974.

[12] A. Atserias. Conjunctive query evaluation by search-tree revisited. *Theor. Comput. Sci.*, 371(3) :155–168, 2007.

[13] W. Le Page B. Goethals and M. Mampaey. Mining interesting sets and rules in relational databases. In *Proceedings of the 25th ACM Symposium on Applied Computing*, 2010.

[14] C. Beeri and P. Buneman, editors. *Database Theory - ICDT '99, 7th International Conference, Jerusalem, Israel, January 10-12, 1999, Proceedings*, volume 1540 of *Lecture Notes in Computer Science*. Springer, 1999.

[15] K. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *Proceedings of the 1999 ACM SIGMOD Conference*, pages 359–370, 1999.

[16] H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, and H. Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16 :135–166, 2002.

[17] P. Bohannon, W. Fan, F. Geerts, X. Jia, and Anastasios Kementsietsidis. Conditional functional dependencies for data cleaning. In *ICDE* [3], pages 746–755.

[18] F. Bonchi and J.-F. Boulicaut, editors. *Knowledge Discovery in Inductive Databases, 4th International Workshop, KDID 2005, Porto, Portugal, October 3, 2005, Revised Selected and Invited Papers*, volume 3933 of *Lecture Notes in Computer Science*. Springer, 2006.

[19] M. Botta, R. Meo, and M. L. Sapino. Incremental execution of the mine rule operator. Technical Report Technical Report RT66-2002, Université de Turin, Turin, May, 2002.

[20] J.-F. Boulicaut, L. De Raedt, and H. Mannila, editors. *Constraint-Based Mining and Inductive Databases, European Workshop on Inductive Databases and Constraint Based Mining, Hinterzarten, Germany, March 11-13, 2004, Revised Selected Papers*, volume 3848 of *Lecture Notes in Computer Science*. Springer, 2005.

[21] S. Brin, R. Motwani, J. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. pages 255–264, In ACM SIGMOD'97, 1997.

[22] S. Brin, R. Rastogi, and K. Shim. Mining optimized gain rules for numeric attributes. *IEEE Transactions on Knowledge and Data Engineering*, 15 :324–338, 2003.

[23] P. G. Bringas, A. Hameurlain, and G. Quirchmayr, editors. *Database and Expert Systems Applications, 21th International Conference, DEXA 2010, Bilbao, Spain, August 30 - September 3, 2010, Proceedings, Part II*, volume 6262 of *Lecture Notes in Computer Science*. Springer, 2010.

[24] S. Ceri, G. Gottlob, and L. Tanca. What you always wanted to know about datalog (and never dared to ask). *IEEE Transactions on Knowledge and Data Engineering*, 1(1) :146–166, March 1989.

[25] W. Chen, J. F. Naughton, and P. A. Bernstein, editors. *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*. ACM, 2000.

[26] C. Chrisment, editor. *19èmes Journées Bases de Données Avancées, BDA '03, 20-23 octobre 2003, Lyon, Actes (Informal Proceedings)*, 2003.

[27] S.S. Cosmadakis, P.C. Kanellakis, and N. Spyratos. Partition semantics for relations. *Journal of Computer and System Sciences*, pages 203–233, 1986.

[28] M. M. Dalkilic and E. L. Roberston. Information dependencies. In *PODS*, pages 245–253, 2000.

[29] L. Dehaspe. *Frequent Pattern Discovery in First-Order-Logic*. PhD thesis, Katholieke Universiteit Leuven, 1998.

[30] L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 125–132. Springer-Verlag, 1997.

[31] L. Dehaspe and H. Toivonen. Discovery of frequent patterns datalog. volume 3, pages 7–36, 1999.

[32] L. Dehaspe and H. Toivonen. Discovery of relational association rules. pages 189–208, 2000.

[33] B. C. Desai, D. Saccà, and S. Greco, editors. *International Database Engineering and Applications Symposium (IDEAS 2009), September 16-18, 2009, Cetraro, Calabria, Italy*, ACM International Conference Proceeding Series. ACM, 2009.

[34] C. T. Dieng, T.-Y. Jen, and D. Laurent. An efficient computation of frequent queries in a star schema. In Bringas et al. [23], pages 225–239.

[35] C. T. Dieng, T.-Y. Jen, and D. Laurent. Frequent query computation in a star schema. *Revue Africaine de la Recherche en Informatique et Mathématiques Appliquées (ARIMA)*, 11 :25–53, Octobre 2010.

[36] C. T. Diop. *Etude et Mise en oeuvre des aspects itératifs de l'extraction de régles d'association dans les bases de données*. PhD thesis, Université de Tours, 2003.

[37] C. T. Diop, A. Giacometti, D. Laurent, and N. Spyratos. Extraction incrémentale de règles d'association par combinaison de tâches d'extraction. In Mouaddib [94].

[38] C. T. Diop, A. Giacometti, D. Laurent, and N. Spyratos. Composition of mining contexts for efficient extraction of association rules. In Jensen et al. [70], pages 106–123.

[39] C. T. Diop, A. Giacometti, D. Laurent, and N. Spyratos. Computation of mining queries : An algebraic approach. In Boulicaut et al. [20], pages 102–126.

[40] C. T. Diop, A. Giacometti, D. Laurent, and N. Spyratos. Extraction itérative de requêtes fréquentes. *Ingénierie des Systèmes d'Information*, 9(3-4) :83–108, 2004.

[41] A. Doucet, editor. *16èmes Journées Bases de Données Avancées, BDA 2000, 24-27 octobre 2000, Blois, Actes (Informal Proceedings)*, 2000.

[42] S. Dzeroski. Inductive logic programming an knowledge discovery in databases. In *In Advances in Knowledge Discovery and Data Mining*, pages 117–152. AAAIMIT Press, 1996.

[43] S. Dzeroski and N. Lavrac. Relational data mining. Berlin, 2001. Springer.

[44] J. F. Elder, F. Fogelman-Soulié, P. A. Flach, and M. Javeed Zaki, editors. *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Paris, France, June 28 - July 1, 2009*. ACM, 2009.

[45] A. Faye, A. Giacometti, D. Laurent, and N. Spyratos. *Mining Rules in Databases with Multiple Tables :Problems and Perspectives*. In 3rd International Conference on Computing Anticipatory Systems (CASYS), Liège, Belgique, 1999.

[46] A. Faye, Arnaud Giacometti, Dominique Laurent, and Nicolas Spyratos. Mining significant rules from databases. *Networking and Information Systems*, 1(6) :653–682, 1998.

[47] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17 :37–54, 1996.

[48] J.-G. Ganascia and P. Gançarski, editors. *Extraction et gestion des connaissances (EGC'2009), Actes, Strasbourg, 27 au 30 janvier 2009*, volume RNTI-E-15 of *Revue des Nouvelles Technologies de l'Information*. Cépaduès-Éditions, 2009.

[49] B. Ganter and R. Wille. Formal concept analysis : Mathematical foundations. Springer-Verlag New York, Secaucus, NJ, USA, Translator-C. Franzke, 1997.

[50] A. Giacometti, D. Laurent, and C. T. Diop. Condensed representations for sets of mining queries. In Meo et al. [91], pages 250–269.

[51] A. Giacometti, D. Laurent, C. T. Diop, and N. Spyratos. La découverte de règles d'association entre vues : vers un processus d'extraction incrémental. In Doucet [41].

[52] A. Giacometti, D. Laurent, and C. Talibouya Diop. Iterative computation of mining queries based on condensed representations. In Chrisment [26].

[53] C. M. Giannella, M. M. Dalkilic, D. P. Groth, and E. L. Robertson. Improving query evaluation with approximate functional dependency based decompositions. In *Proceedings of the 19th British National Conference on Databases (BNCOD2002)*, 2002.

[54] B. Goethals and J. Van den Bussche. Relational association rules : Getting warmer. In David Hand, Niall Adams, and Richard Bolton, editors, *Pattern Detection and Discovery*, volume 2447 of *Lecture Notes in Computer Science*, pages 145–159. Springer Berlin,Heidelberg, 2002.

[55] B. Goethals, D. Laurent, and W. Le Page. Discovery and application of functional dependencies in conjunctive query mining. In *Proceedings of DaWak 2010*, 2010.

[56] B. Goethals, W. Le Page, and Heikki Mannila. Mining association rules of simple conjunctive queries. In *SDM* [4], pages 96–107.

[57] R. L. Grossman, J. Han, V. Kumar, H. Mannila, and R. Motwani, editors. *Proceedings of the Second SIAM International Conference on Data Mining, Arlington, VA, USA, April 11-13, 2002*. SIAM, 2002.

[58] J. Ullman H. Garcia-Molina and J. Widom. Database system implementation. Seattle, Washington, 2001. Prentice-Hall.

[59] Jiawei Hah, Yongjian Fu, Wei Wang, Krzysztof Koperski, and Osmar Zaiane. Dmql : A data mining query language for relational databases. In *Research Issues on Data Mining and Knowledge Discovery*.

[60] J. Han and M. Kamber. Datamining concepts and technics. Morgann Kaufmann, San Francisco, 1992.

[61] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation : A frequent-pattern tree approach. *Data Min. Knowl. Discov.*, 8(1) :53–87, 2004.

[62] Y. Huhtala, J. Karkkainen, P. Porkka, and H. Toivonen. Efficient discovery of functional and approximate dependencies using partitions. In *ICDE* [1], pages 392–401.

[63] T. Imielinski and H. Mannila. A database perspective on knowdledge discovery. In *communication of the ACM*, volume 39, pages 58–64, November 1996.

[64] M. Jarke, J. Clifford, and Y. Vassiliou. An optimizing prolog front-end to a relational query system. pages 296–306, 1984.

[65] T.-Y. Jen, D. Laurent, and N. Spyratos. Mining all frequent projection-selection queries from a relational table. In Kemper et al. [73], pages 368–379.

[66] T.-Y. Jen, D. Laurent, and N. Spyratos. Mining frequent conjunctive queries in star schemas. In Desai et al. [33], pages 97–108.

[67] T.-Y. Jen, D. Laurent, and N. Spyratos. Computing supports of conjunctive queries on relational tables with functional dependencies. *Fundam. Inform.*, 99(3) :263–292, 2010.

[68] T.-Y. Jen, D. Laurent, N. Spyratos, and O. Sy. Mining frequent queries in star schemes. In Pinson and Vincent [103], pages 331–342.

[69] T.-Y. Jen, D. Laurent, N. Spyratos, and O. Sy. Towards mining frequent queries in star schemes. In Bonchi and Boulicaut [18], pages 104–123.

[70] C. S. Jensen, K. G. Jeffery, J. Pokorný, S. Saltenis, E. Bertino, K. Bohm, and M. Jarke, editors. *Advances in Database Technology - EDBT 2002, 8th International Conference on Extending Database Technology, Prague, Czech Republic, March 25-27, Proceedings*, volume 2287 of *Lecture Notes in Computer Science*. Springer, 2002.

[71] D. S. Johnson and A. C. Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. *J. Comput. Syst. Sci.*, 28(1) :167–189, 1984.

[72] K. Jyrki and H. Mannila. The power of sampling in knowledge discovery. In *Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '94, pages 77–85, New York, NY, USA, 1994. ACM.

[73] A. Kemper, P. Valduriez, N. Mouaddib, J. Teubner, M. Bouzeghoub, V. Markl, L. Amsaleg, and I. Manolescu, editors. *EDBT 2008, 11th International Conference on Extending Database Technology, Nantes, France, March 25-29, 2008, Proceedings*, volume 261 of *ACM International Conference Proceeding Series*. ACM, 2008.

[74] R. Kenneth and D. Srivastava. Fast computation of sparse datacubes. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, VLDB '97, pages 116–125, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.

[75] M. Klemettinen and R. Meo, editors. *Proceedings of the First International Workshop on Inductive Databases, 20 August 2002, Helsinki, Finland*. Helsinki University Printing House, Helsinki, 2002.

[76] Y. Kodratoff. Research topics in knowledge discovery in data and texts. In 3rd SIGMOD 98 Workshop on Research Issues in Data Mining and Knowledge Discovery, Seattle, WA, 1998.

[77] Y. Kodratoff. Rating the interest of rules induced from data and within texts. In 12th IEEE- International Conference on Database and Expert Systems Applications, Munich, 2001 2001.

[78] A. Koopman and A. Siebes. Characteristic relational patterns. In Elder et al. [44], pages 437–446.

[79] M. Kuramochi and G. Karypis. Frequent subgraph discovery. *Data Mining, IEEE International Conference on*, 0 :313, 2001.

[80] D. Laurent. Relational databases. In Wah [118].

[81] N. Lavrac, P. Flash, and B. Zupan. Rule evaluation measures : A unifying view. In *In 9th Intl. Workshop ILP'99*, volume 1297. Bled, Slovnia, June 1999.

[82] L.Copin, N. Pecheur, A. Laurent, Y.Augusta, B. Sentana, D. Laurent, and T.-Y. Jen. Dbfrequentqueries : Extraction de requêtes fréquentes. In Ganascia and Gançarski [48], page 499.

[83] M. Levene and M. W. Vincent. Justification for inclusion dependency normal form. *IEEE Transactions on Knowledge and Data Engineering*, 12 :281–291, 2000.

[84] J. Looyd. Foundations of logic programming. Springer-Verlag, second extended edition, 1987.

[85] M. Ogihara M. J. Zaki, S. Parthasarathy and W. Li. New algorithms for fast discovery of association rules. Technical Report TR651, 1997.

[86] R. Cichetti M. Laporte, N. Novelli and L. Lakhal. Computing full and iceberg datacubes using partitions. ISMIS, Lyon, France, June 27-29 2002.

[87] H. Mannila. Inductive databases and condensed representations for data mining. In *ILPS'97*, pages 21–30, 1997.

[88] H. Mannila. Theoretical frameworks for data mining. *SIGKDD Explor. Newsl.*, 1 :30–32, January 2000.

[89] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. Technical Report C-1997-8, University of Helsinki, Helsinki, 1997.

[90] H. Mannila, H. Toivonen, and A. Inkeri Verkamo. Efficient algorithms for discovering association rules. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *AAAI, Workshop on Knowledge Discovery in Databases (KDD-94)*, pages 181–192, Seattle, Washington, 1994. AAAI Press.

[91] R. Meo, P. L. Lanzi, and M. Klemettinen, editors. *Database Support for Data Mining Applications : Discovering Knowledge with Inductive Queries*, volume 2682 of *Lecture Notes in Computer Science*. Springer, 2004.

[92] R. Meo, G. Psaila, and S. Ceri. An extension to sql for mining association rules. *Data Mining Knowledge Discovery*, 2 :195–224, June 1998.

[93] T. M. Mitchell. Generalization as search. *Artif. Intell.*, 18(2) :203–226, 1982.

[94] N. Mouaddib, editor. *17èmes Journées Bases de Données Avancées, BDA 2001, 29 octobre - 2 novembre, Agadir (Maroc), Actes (Informal Proceedings)*, 2001.

[95] S. Muggleton. Inverse entailment and progol. In *New generation Computing, Special Issue on Logic Programming*, 1995.

[96] E. Ka Ka Ng, Ka Ng, A. Wai-Chee Fu, and K. Wang. Mining association rules from stars. In *In Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)*, pages 322–329, 2002.

[97] S. Nijssen and J. N. Kok. Faster association rules for multiple relations. In *International Joint Conference on Artificial Intelligence*, pages 891–896. Morgan Kaufmann, 2001.

[98] S. Nijssen and J. N. Kok. Efficient frequent query discovery in farmer. In *In Proc. of the 7th PKDD, volume 2838 of Lecture Note in Computer Science*, pages 350–362. Springer, 2003.

[99] W. Le Page. *Mining Pattern in Relational Databases*. PhD thesis, University of Antwerpen, 2009.

[100] J. S. Park, M. S. Chen, and P. S. Yu. An effective hash-based algorithm for mining association rules. In *SIGMOD '95 : Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, pages 175–186, New York, NY, USA, 1995. ACM.

[101] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Discovering frequent closed itemsets for association rules. In Beeri and Buneman [14], pages 398–416.

[102] J. Pei, J. Han, and R. Mao. Closet : An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, pages 21–30, 2000.

[103] S. Pinson and N. Vincent, editors. *Extraction et gestion des connaissances (EGC'2005), Actes des cinquièmes journées Extraction et Gestion des Connaissances, Paris, France, 18-21 janvier 2005, 2 Volumes*, volume RNTI-E-3 of *Revue des Nouvelles Technologies de l'Information*. Cépaduès-Éditions, 2005.

[104] G.D. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[105] M. Protasi and R. Fagin. Acyclic database schemes (of various degrees) : A painless introduction. In *Research report rj3800, IBM Almaden Research*, 1983.

[106] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. In Chen et al. [25], pages 249–260.

[107] R. de Wolf S. H. Nienhuys-Cheng. Fondations of inductive logic programming. volume 1228 of Lectures Notes in Artificial Intelligence. Springer-Verlag, 1997.

[108] S. Sarawagi, R. Agrawal, and A. Gupta. On computing the datacube. Technical Report Technical Report RJ10026, IBM Almaden Research Center, San Jose, CA, 1996.

[109] S. Sarawagi, S. Thomas, and R. Agrawal. Integrating association rule mining with relational database systems : Alternatives and implications. *Data Min. Knowl. Discov.*, 4(2/3) :89–125, 2000.

[110] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In Umeshwar et al. [117], pages 432–444.

[111] T. K. Sellis. Multiple-query optimization. *ACM Trans on Database Syst.*, 13 :23–52, 1988.

[112] S.Lopes, J.-M. Petit, and L. Lakhal. Efficient discovery of functional dependencies and armstrong relations. In Zaniolo et al. [122], pages 350–364.

[113] N. Spyratos. The partition model : A deductive database model. *ACM Trans. Database Syst.*, 12(1) :1–37, 1987.

[114] H. Toivonen. Sampling large databases for association rules. pages 134–145. Morgan Kaufmann, 1996.

[115] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume I.* Computer Science Press, 1988.

[116] J. D. Ullman. *Principles of Database and Knowledge-Base Systems, Volume II.* Computer Science Press, 1989.

[117] D. Umeshwar, P. M. D. Gray, and S. Nishio, editors. *VLDB'95, Proceedings of 21th International Conference on Very Large Data Bases, September 11-15, 1995, Zurich, Switzerland*. Morgan Kaufmann, 1995.

[118] B.W. Wah, editor. *Wiley Encyclopedia of Computer Science and Engineering*. John Wiley & Sons, Inc., 2008.

[119] E.W Weisstein. Restricted growth string. In *MathWorld-A Wolfram Web Resource*, pages 487–499, 2009.

[120] P. S. Yu and A. L. P. Chen, editors. *Proceedings of the Eleventh International Conference on Data Engineering, March 6-10, 1995, Taipei, Taiwan*. IEEE Computer Society, 1995.

[121] M. J. Zaki and C.-J. Hsiao. Charm : An efficient algorithm for closed itemset mining. In Grossman et al. [57].

[122] C. Zaniolo, P. C. Lockemann, M. H. Scholl, and Torsten Grust, editors. *Advances in Database Technology - EDBT 2000, 7th International Conference on Extending Database Technology, Konstanz, Germany, March 27-31, 2000, Proceedings*, volume 1777 of *Lecture Notes in Computer Science*. Springer, 2000.

Table des figures

1.1 Base de données introductive 4

2.1 Le Data Mining : une partie du processus global d'extraction de motifs intéressants. 10
2.2 Exemple d'une table de transactions (1) 14
2.3 Exemple d'une table transactionnelle (2) 14
2.4 Table transactionnelle sous forme d'une table relationnelle 14
2.5 Représentation Classique d'une table relationnelle 15
2.6 Représentation d'une table relationnelle sous forme Transactionnelle 15
2.7 Base de données des buveurs 26
2.8 Espace de recherche des blocs pour 4 attributs 30
2.9 Caractéristiques des bases IMDB et QuizDB 37
2.10 Evolution du temps en fonction du support pour ImDB 38
2.11 Evolution du temps en fonction du support pour QuizDB 38

3.1 Base de données de référence 47
3.2 La table J et la table auxiliaire associée AUX(J) en considérant l'Exemple 3.1 69

4.1 Interface Graphique de FQF 80
4.2 Architecture globale de notre application de recherche de requêtes fréquentes (FQF) 81
4.3 Redondance de Projection (proposition 3.6(2)) 86
4.4 Redondance de Sélection (proposition 3.6(3)) 86
4.5 Redondance de Projection-Sélection (proposition 3.6(4)) 86
4.6 Redondance induite par l'application de proposition 3.6(3) puis proposition 3.6(2) et proposition 3.6(2) puis proposition 3.6(3) 87
4.7 Redondance induite par l'application de proposition 3.6(3) puis proposition 3.6(2) et proposition 3.6(2), proposition 3.6(2) puis proposition 3.6(4) . . . 87
4.8 Comparaison de notre implémentation à celle précédente 96
4.9 Temps d'exécution de FQF en fonction du nombre de tuples dans la table de faits. 97
4.10 Temps d'exécution de FQF en fonction du nombre de dimensions. 97
4.11 Temps d'exécution de FQF en fonction du nombre de mesures. 98
4.12 Temps d'exécution de FQF en fonction du nombre de tuples de la table de faits pour de petites bases. 98
4.13 Temps d'exécution de FQF en fonction du nombre de tuples de la table de faits pour de grandes bases. 99
4.14 Temps d'exécution en fonction du seuil de support. 99
4.15 Comparaison du temps d'exécution de `FQF`, `Conqueror`$^+$ et `Conqueror` en fonction du seuil de support 100
4.16 Comparaison du temps d'exécution de `FQF`, `Conqueror`$^+$ et `Conqueror`s en fonction du nombre de dimensions 100

4.17 Comparaison du temps d'exécution de FQF et Conqueror en fonction du seuil de support 101

B.1 Diagramme de Classes 124
B.2 Diagramme de séquences de la fonction generate() 126
B.3 Diagrammme de séquences de la fonction prune() 127
B.4 Diagrammme de séquences de la fonction scan() 128

Appendices

A

Preuves des propositions

La proposition suivante permet de caractériser $succ(q)$ dans le cas d'une table relationnelle.

A.1 Proposition

Soient une table r et une classe $[q]$ ayant pour représentant la requête $\pi_X\sigma_y(r)$ de $\mathcal{C}(\Delta)$ différente de C_1, $succ(q)$ est égale à l'ensemble des classes obtenues de la manière suivante :

1. *$succ(q) = \{C_1\}$ si et seulement, pour chaque $Y' \in Y^{\uparrow}$, $X \subseteq Y'$.*
2. *Pour chaque schéma non vide Z, si*
   (a) $Y \subset (X \setminus Z)$, et
   (b) $(X \setminus Z) \in X^{\downarrow}$,
   alors $succ(q)$ contient la classe $\pi_{X'}\sigma_{y'}$ avec $X' = (X \setminus Z)$, $Y' = Y$ et $y' = y$.
3. *Pour tout tuple z sur schéma non vide Z tel que $yz \in \pi_{YZ}(\Delta)$, si*
   (a) $Z \subseteq X$,
   (b) $YZ \in Y^{\uparrow} \setminus \{X\}$, et
   (c) tel qu'il n'existe pas de schéma X_0 de $cl(FD) \setminus \{\emptyset\}$ tel que $Y \subseteq X_0 \subset X \subseteq (X_0Z)^+$,
   alors $succ(q)$ contient la classe $\pi_{X'}\sigma_{y'}$ avec $X' = X$, $Y' = YZ$ et $y' = yz$.
4. *Pour chaque tuple z sur un schéma non vide Z tel que $yz \in \pi_{YZ}(\Delta)$, si*
   (a) $Z \not\subseteq X$,
   (b) $YZ \in Y^{\uparrow}$, et
   (c) tel qu'il n'existe pas de schéma X_0 de $cl(FD)$ tel que $Y \subseteq X_0 \subset X$ et $(X_0Z)^+ = (XZ)^+$,
   alors $succ(q)$ contient la classe $\pi_{X'}\sigma_{y'}$ avec $X' = (XZ)^+$, $Y' = YZ$ et $y' = yz$.

Démonstration. En ce qui concerne la preuve de l'item 1 de la proposition, nous notons que, comme C_1 est la classe la plus spécifique, si $C_1 \in succ(\pi_X\sigma_y)$, alors $succ(\pi_X\sigma_y) = \{C_1\}$.

Supposons tout d'abord que, pour chaque $Y' \in Y^{\uparrow}$, $X \subseteq Y'$ et qu'il existe une requête classe de requête ayant pour représentant $\pi_{X''}\sigma_{y''}$ vérifiant $\pi_X\sigma_y(r) \prec \pi_{X''}\sigma_{y''}(r) \prec C_1$. Comme la comparaison est stricte, $Y \rightarrow X$ et $Y'' \rightarrow X''$ ne sont pas dans FD^+.

Ainsi, $Y \subset X$, $Y'' \subset X''$, $X'' \subseteq (XY'')^+$, $Y \subseteq Y''$, $y''.Y = y$, et l'une des deux dernières inclusions est stricte.
– Si $Y = Y''$ alors $X'' \subset (XY'')^+ = X$. Comme dans ce cas, $Y \subset X''$, il existe $Y' \in Y^\uparrow$ tel que $Y \subset Y' \subseteq X''$. Ainsi, $X \subseteq Y' \subseteq X''$, ce qui est en contradiction avec $X'' \subset X$.
– Si $Y \neq Y''$, alors il existe $Y' \in Y^\uparrow$ tel que $Y \subset Y' \subseteq Y''$. Ainsi, $X \subseteq Y' \subset Y''$, donc $X'' \subseteq (XY'')^+ = Y''$, ce qui est impossible.

Ainsi, nous montrons que pour tout $Y' \in Y^\uparrow$, $X \subseteq Y'$, $succ(\pi_X\sigma_y) = \{C_1\}$.

Inversement, soit une classe ayant pour représentant $\pi_X\sigma_y(r)$ tel que $succ(\pi_X\sigma_y(r)) = \{C_1\}$ et supposons qu'il existe Y' appartenant à $Y^\uparrow$ tel que $X \not\subseteq Y'$. Alors dans ce cas, $X \neq Y'$ et :
– Si $Y' \subset X$ alors, pour tout y' de $\pi_{Y'}(\Delta)$ tel que $y'.Y = y$, nous $\pi_X\sigma_y(r) \prec \pi_X\sigma_{y'}(r) \prec C_1$, et ainsi, $C_1 \notin succ(\pi_X\sigma_y)$.
– Si $Y' \not\subset X$ alors $Y' \subset XY' \subseteq (XY')^+$. Dans ce cas, en considérant le tuple y' défini plus haut, nous avons $\pi_X\sigma_y \prec \pi_{(XY')^+}\sigma_y \prec C_1$, et ainsi, $C_1 \notin succ(\pi_X\sigma_y(r))$.
De ce fait, l'item 1 de la proposition est vérifié

Pour démontrer les autres items, supposons que $succ(\pi_X\sigma_y(r)) \neq \{C_1\}$, nous notons par $\Sigma(\pi_X\sigma_y(r))$ l'ensemble de toutes les classes défini par les trois derniers items de la proposition, montrons que $\Sigma(\pi_X\sigma_y(r)) = succ(\pi_X\sigma_y)$.

Supposons qu'il existe une classe $\pi_{X''}\sigma_{y''}(r)$ telle que $\pi_X\sigma_y(r) \prec \pi_{X''}\sigma_{y''}(r) \prec \pi_{X'}\sigma_{y'}(r)$, nous allons considérer successivement les trois derniers items de la proposition.
Ainsi, nous avons $X'' \subseteq (XY'')^+$, $X' \subseteq (X''Y')^+$, $Y \subseteq Y''$ et $Y'' \subseteq Y'$.
2. Si $Y = Y'$, alors $Y = Y' = Y''$. Ainsi, $X'' \subset X$ et $X \setminus Z \subset X''$, ce qui est en contradiction avec la définition de Z.
3. Si $X = X'$ et $Y' = YZ$, alors $X'' \subseteq (XY'')^+$, $X \subseteq (X''YZ)^+$ et $Y \subseteq Y'' \subseteq YZ$. De ce fait, $X \subseteq (XY''YZ)^+ = (XY'')^+$. De plus, comme $Y'' \in cl(FD)$, grâce à l'item 3(b), nous avons soit $Y'' = Y$ soit $Y'' = Y' = YZ$.
Si $Y'' = Y$ alors $X'' \subset X \subseteq (X''YZ)^+ = (X''Z)^+$. Ainsi, $Y'' \subseteq X'' \subset X \subseteq (X''Z)^+$, ce qui est en contraction avec l'item 3(c).
Si $Y'' = Y' = YZ$ alors $X'' \subseteq (XZ)^+ = X$ et $X \subset (X''YZ)^+ = (X''Y'')^+ = X''$. Ainsi, $X \subset X'' \subseteq X$, ce qui est impossible.
4. Si $X' = (XZ)^+$ et $Y' = YZ$, alors $Y \subseteq Y'' \subseteq YZ$ et ainsi, en utilisant l'item 4(b), nous avons $Y'' = Y$ ou $Y'' = Y' = YZ$.

Si $Y'' = Y$ alors $X'' \subset (XY)^+ = X$ et $(XZ)^+ \subseteq (X''YZ)^+ = (X''Z)^+$. Ainsi, $Y \subseteq X'' \subset X$ et $(XZ)^+ \subseteq (XZ)^+$. Comme $X'' \subset X$, nous avons $(X''Z)^+ \subseteq (XZ)^+$. Nous en déduisons que, $(X''Z)^+ = (XZ)^+$, ce qui est en contraction avec l'item 4(c).

Si $Y'' = Y' = YZ$ alors $X'' \subseteq (XZ)^+$, $YZ \subseteq X''$ et $(XZ)^+ \subset (X''YZ)^+ = X''$. Nous en déduisons que, $X'' \subseteq (XZ)^+ \subset X''$, ce qui est impossible
Nous avons ainsi montré que $\Sigma(\pi_X\sigma_y(r)) \subseteq succ(\pi_X\sigma_y(r))$.
Nous allons maintenant montrer l'inclusion dans l'autre sens, *i.e.*, $succ(\pi_X\sigma_y(r)) \subseteq \Sigma(\pi_X\sigma_y(r))$.
Soit $\pi_{X'}\sigma_{y'} \in succ(\pi_X\sigma_y(r))$, alors nous anons $\pi_X\sigma_y(r) \prec \pi_{X'}\sigma_{y'}(r)$ et il n'existe pas de classe $\pi_{X''}\sigma_{y''}$ de $\mathcal{C}(\Delta)$ telle que $\pi_X\sigma_y \prec \pi_{X''}\sigma_{y''} \prec \pi_{X'}\sigma_{y'}$. Ainsi, $X' \subseteq (XY')^+$, $Y \subseteq Y'$, $y = \Delta_Y(y')$ et au moins une des deux inclusions est stricte.
Si $y = y'$, alors, si $Y = Y'$, alors $X' \subset (XY')^+ = X$. Ainsi X' peut être écrit sous la forme $X \setminus Z$, où $Y \subseteq X \setminus Z$ et $(X \setminus Z) \in cl(FD)$.

Ainsi, l'item 2(a) dans la définition de $\Sigma(\pi_X\sigma_y(r))$ est vérifié.
Supposons que l'item 2(b) ne soit pas satisfait, *i.e.,* alors il existe X_0 de $cl(FD)$ tel que $(X \setminus Z) \subset X_0 \subset X$.
Alors, nous avons $\pi_X\sigma_y(r) \prec \pi_{X_0}\sigma_y(r)$ et $\pi_{X_0}\sigma_y \prec \pi_{X'}\sigma_y(r)$, ce qui est en contradiction avec le fait que $\pi_X\sigma_y(r) \in succ(\pi_X\sigma_y(r))$.
Ainsi, dans ce cas $\pi_{X'}\sigma_{y'}()$ satisfait l'item 2 de la définition de $\Sigma(\pi_X\sigma_y)$, ce qui montre que $\pi_{X'}\sigma_{y'}(r) \in \Sigma(\pi_X\sigma_y(r))$.

Si $y \neq y'$, alors $Y \subset Y'$, ainsi, $y' = yz$ avec Z est tel que $Y' = YZ$ et $YZ \in cl(FD)$.
Nous considérons deux cas :
– YZ est un sous ensemble de X,
– YZ n'est pas un sous ensemble de X.

– Supposons tout d'abord que $YZ \subseteq X$. Notons que si $X = YZ$ alors, comme $\pi_X\sigma_y(r) \prec \pi_{X'}\sigma_{y'}(r)$, $X' \subseteq (XY')^+$ peut être écrite sous la forme $X' \subseteq YZ$, ce qui signifie que $X' \subseteq Y'$. ce qui n'est pas possible du moment que $\pi_{X'}\sigma_{y'}(r) \neq C_1$. Ainsi, $X \neq YZ$.

S'il existe Z' tel que $YZ' \in cl(FD)$ et $Y \subset YZ' \subset YZ$, alors nous avons $\pi_X\sigma_y(r) \prec \pi_X\sigma_{yz'}(r) \prec \pi_{X'}\sigma_{yz}(r)$, avec $yz' = yz.(YZ')$. En effet, d'une part, $X \subseteq (XYZ')^+$, $Y \subset YZ'$ et $y = \Delta_Y(yz')$, et d'autre part, $X' \subseteq (XYZ)^+$, $YZ' \subset YZ$ et $yz' = \Delta_{YZ'}(yz)$.
Ce cas est impossible puisque $\pi_{X'}\sigma_{y'}(r) \in succ(\pi_X\sigma_y)(r)$, et ainsi l'item 3(b) de la définition de $\Sigma(\pi_X\sigma_y(r))$ est vérifié.

S'il existe X_0 de $cl(FD) \setminus \{\emptyset\}$ tel que $Y \subseteq X_0 \subset X \subseteq (X_0Z)^+$, alors nous avons $\pi_X\sigma_y(r) \prec \pi_{X_0}\sigma_y(r) \prec \pi_{X'}\sigma_{yz}(r)$.
En effet, $X_0 \subset (XY)^+ = X$, $X' \subseteq (X_0YZ)^+$ (car $X' \subseteq (XYZ)^+$ et $YZ \subseteq X$ implique que $X' \subseteq X$, et $X \subseteq (X_0Z)^+ \subseteq (X_0YZ)^+$), $YZ \subset Y$ et $y = \Delta_Y(yz)$.
Ce cas est aussi impossible comme $\pi_{X'}\sigma_{y'}(r)$ est dans $succ(\pi_X\sigma_y(r))$, et ainsi l'item 3(c) de la définition de $\Sigma(\pi_X\sigma_y(r))$ est vérifié.
Ainsi, l'item 3 de la définition de $\Sigma(\pi_X\sigma_y(r))$ est vérifié, ce qui montre que $\pi_{X'}\sigma_{y'}(r) \in \Sigma(\pi_X\sigma_y(r))$.

– Supposons maintenant que $YZ \not\subseteq X$. S'il existe Z' tel que $YZ' \in cl(FD)$ et $Y \subset YZ' \subset YZ$, alors nous avons $\pi_X\sigma_y(r) \prec \langle (XZ')^+, yz' \rangle \prec \pi_{X'}\sigma_{yz}(r)$, avec $yz' = yz.(YZ')$. En effet, nous avons d'une part $(XZ')^+ \subseteq (XYZ')^+$ (car, comme $Y \subseteq X$, $XZ' = XYZ'$), $Y \subset YZ'$ et $y = \Delta_Y(yz')$, et d'autre part, $X' \subseteq (XYZZ')^+$ (car, comme $YZ' \subseteq YZ$ et $Y \subseteq X$, $XYZZ' = XZ$ et $X' \subseteq (XY')^+ = (XYZ)^+ = (XZ)^+$), $YZ' \subset YZ$ et $yz' = \Delta_{YZ'}(yz)$.
Ce cas est impossible comme $\pi_{X'}\sigma_{y'}(r)$ est dans $succ(\pi_X\sigma_y(r))$, et ainsi l'item 4(b) de la définition de $\Sigma(\pi_X\sigma_y(r))$ est vérifié.

S'il existe $X_0 \in cl(FD)$ tel que $Y \subseteq X_0 \subset X$ et $(XZ)^+ = (X_0Z)^+$, alors $\pi_X\sigma_y(r) \prec \pi_{X_0}\sigma_y(r) \prec \pi_{X'}\sigma_{yz}(r)$.
En effet, nous avons d'une part $X_0 \subset (XY)^+$, et d'autre part $X' \subseteq (X_0YZ)^+$ (car $X' \subseteq (XY')^+$), $Y' = YZ$, $Y \subseteq X$, $(X_0Z)^+ = (XZ)^+$ et $Y \subseteq X_0$ implique que $X' \subseteq (X_0Z)^+ = (X_0YZ)^+$), $YZ \subset YZ$ et $y = \Delta_Y(yz)$.
Ce cas est impossible comme $\pi_{X'}\sigma_{y'}(r)$ est dans $succ(\pi_X\sigma_y(r))$, et ainsi item 4(c) de la définition de $\Sigma(\pi_X\sigma_y(r))$ est vérifié.
Ainsi, l'item 4 de la définition de $\Sigma(\pi_X\sigma_y)$ est vérifié, ce qui montre que $\pi_{X'}, \sigma_{y'}(r) \in \Sigma(\pi_X\sigma_y(r))$. Nous en déduisons que, $succ(\pi_X\sigma_y) \subseteq \Sigma(\pi_X\sigma_y)$, ce qui complète la preuve. □

A.2 Preuve proposition 3.7

Soit une classe $[q]$ *de requêtes ayant pour représentant* $q = \pi_X \sigma_y(r)$ *:* $\{succ(q) : q \in \langle X, Y, r\rangle\} = succ(\langle X, Y, r\rangle)$.

Démonstration. Pour l'égalité de ces deux ensembles, montrons que : $\{succ(q) : q \in \langle X, Y, r\rangle\} \subseteq succ(\langle X, Y, r\rangle)$ puis que $succ(\langle X, Y, r\rangle) \subseteq \{succ(q) : q \in \langle X, Y, r\rangle\}$.

- Soit $q' = \pi_{X'}\sigma_{y'} \in \{succ(q) : q \in \langle X, Y, r\rangle\}$, ceci implique que q' vérifie l'une des 3 items suivants d'après la Proposition 3.6 :
  1. $Y_R^{\uparrow} = \{X\}$ ce qui implique que que q' appartient bien à C_1 d'après la Définition 3.7.(1), d'où $q' \in succ(\langle X, Y, r\rangle)$ ou
  2. $q' = \pi_{X'}\sigma_{y'}(r)$ où $X' = (X \setminus A)$ et $y = y'$, avec $A \in (X \setminus Y)$ est tel que $Y \subset (X \setminus A)$ et $(X \setminus A) \in X_R^{\downarrow}$, ainsi, $q \in succ(\langle X, Y, r\rangle)$.
  3. (a) $q' = \pi_{X'}\sigma_{y'}(r)$, avec $X' = R$ et $Y' = YA$, si $A \in K$, $(K \setminus A) \subseteq X$ et $(M \cap (X \setminus Y)) = \emptyset$, ce qui implique que $q' \in succ(\langle X, Y, r\rangle)$ d'après la Définition 3.7.3.(a).

     (b) $q' = \pi_{X'}\sigma_{y'}(r)$, avec $X' = XA$ et $Y' = YA$, ($A \in K$ et $(K \setminus A) \not\subseteq X$ et $XA \in X_R^{\uparrow}$) ou $(A \notin K)$, alors ce qui implique que $q' \in succ(\langle X, Y, r\rangle)$ d'après la Définition 3.7.3.(b).
- Soit $q' \in succ(\langle X, Y, r\rangle)$:
  1. $Y \to X$, ceci implique que $q' \in C_1$, d'où $q' \in \{succ(q) : q \in \langle X, Y, r\rangle, Y \to X\} = C_1$
  2. q' est de la forme $q' = \pi_{X\setminus A}\sigma_y(r)$, où $A \in (X \setminus Y)$ et $Y \subset (X \setminus A)$ et $(X \setminus A) \in X_R^{\downarrow}$ et $y' = t.Y$ (où t est un tuple de r), ainsi d'après la Proposition 3.6, $q' \in succ(q)$ avec $q = \pi_X \sigma_y(r)$, où $y = y'$, qui appartient bien à $\langle X, Y, r\rangle$ d'après la Définition 3.7.(2), d'où, $q' \in \{succ(q) : q \in \langle X, Y, r\rangle\}$, ou,
  3. q' est de la forme $q' = \pi_{X'}\sigma_{y'}(r)$ avec : Si A est tel que $A \in (R \setminus X)$, $YA \in Y_R^{\uparrow}$ et y' est tout tuple sur Y' tel que $y' = ya$ et $ya \in \pi_{YA}(r)$, alors
     (a) si $A \in K$, $(K \setminus A) \subseteq X$ et $(M \cap (X \setminus Y)) = \emptyset$, alors $X' = R$ et $Y' = YA$, ceci implique d'après la Proposition 3.6.(3).(a), que $q' \in succ(q)$ avec $q = \pi_X \sigma_y(r)$ qui appartient bien à $\langle X, Y, r\rangle$.

     (b) si ($A \in K$ et $(K \setminus A) \not\subseteq X$ et $XA \in X_R^{\uparrow}$) ou $(A \notin K)$, alors $X' = XA$ et $Y' = YA$, ceci implique d'après la Proposition 3.6.(3).(a), que $q' \in succ(q)$ avec $q = \pi_X \sigma_y(r)$ qui appartient bien à $\langle X, Y, r\rangle$.

□

B

Documentation des classes Java

B.1 Diagrammes de classes

Le diagramme suivant illustre les dépendances entre les différentes classes de notre application :

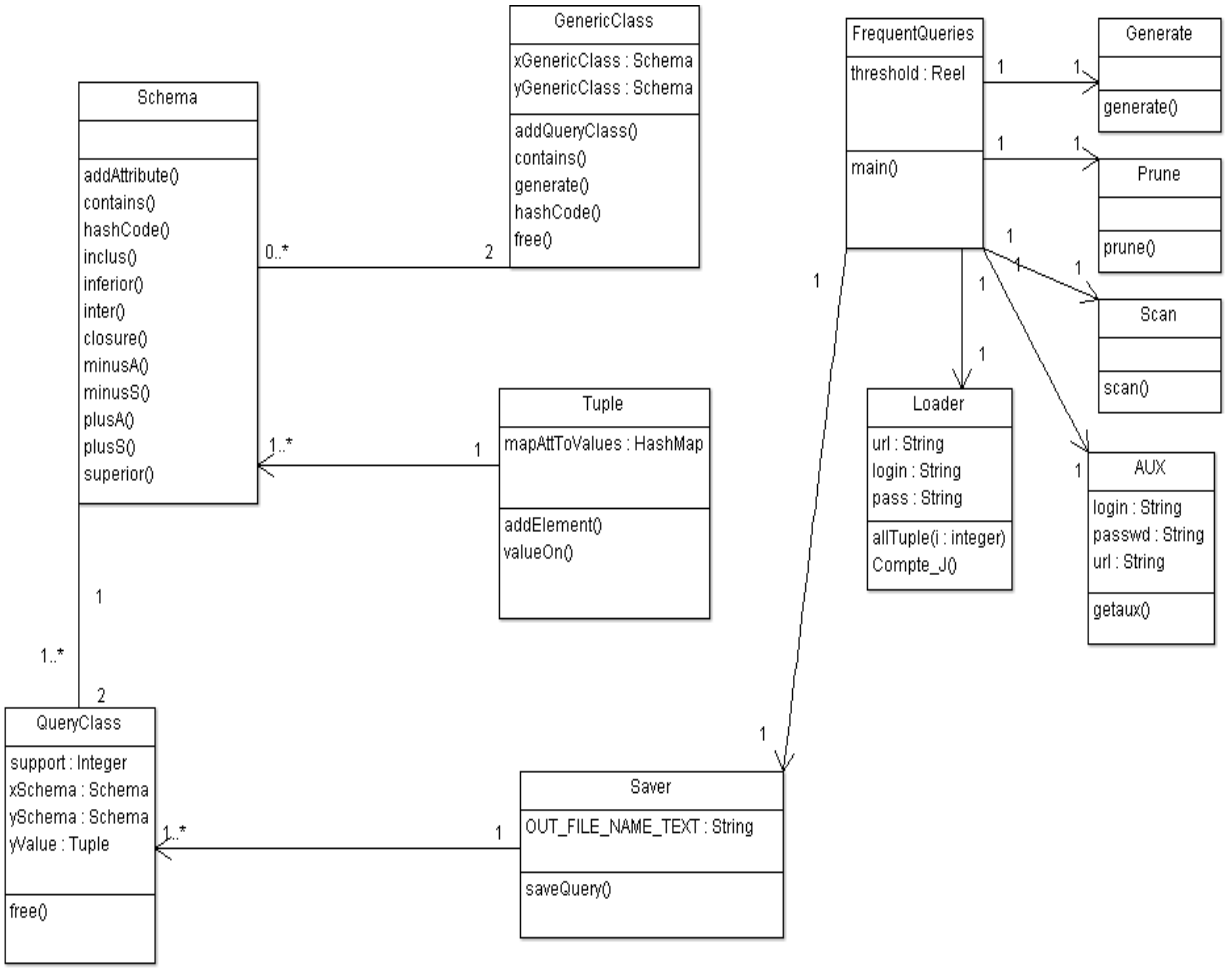


FIGURE B.1 – Diagramme de Classes

B.2 Diagrammes de séquences

Les diagrammes de séquences suivant montrent en détail les différentes interactions entre les différents objets du système, les instanciations, les appels de fonctions ainsi que les destructions d'objets.

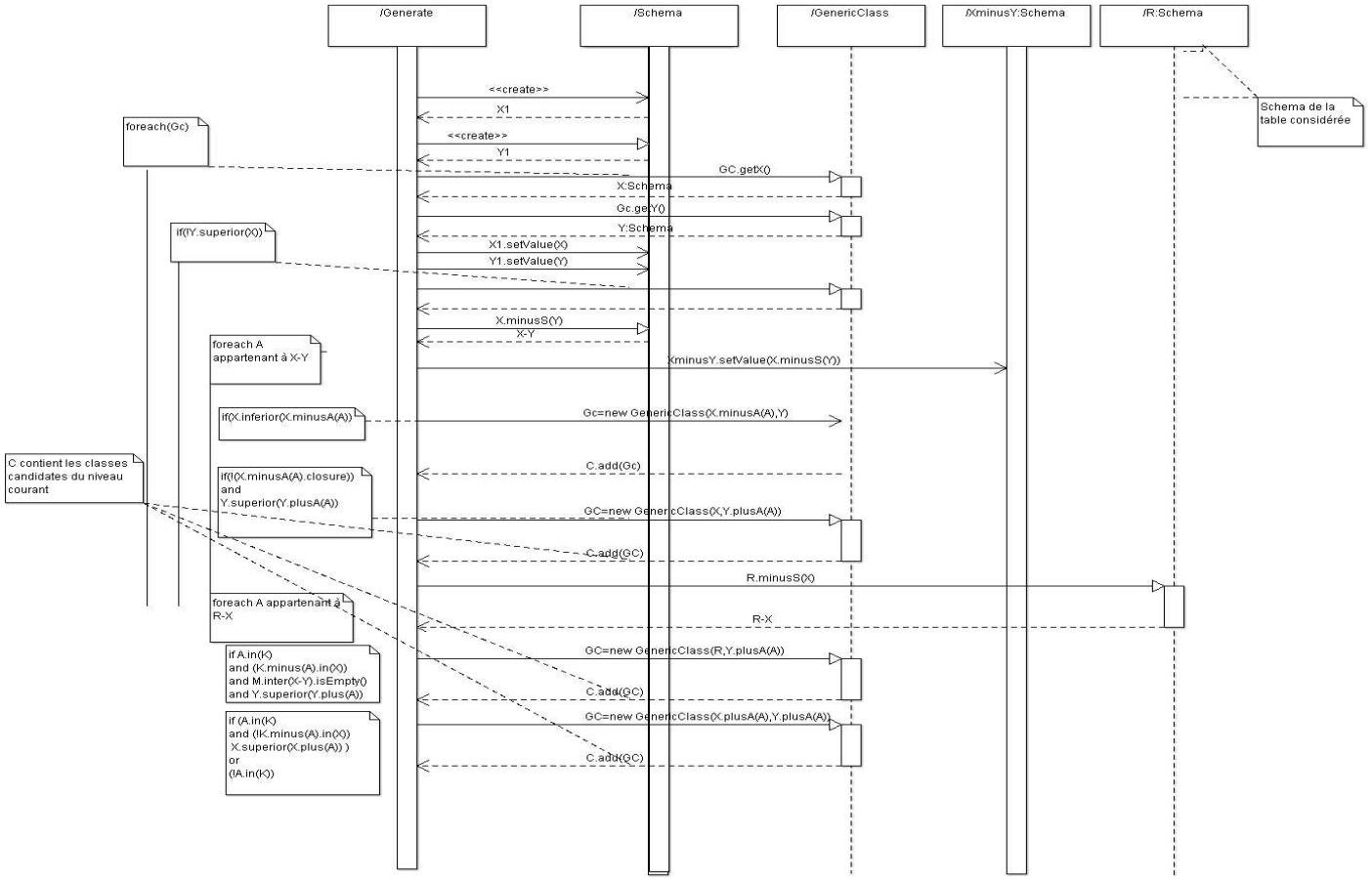


FIGURE B.2 – Diagramme de séquences de la fonction generate()

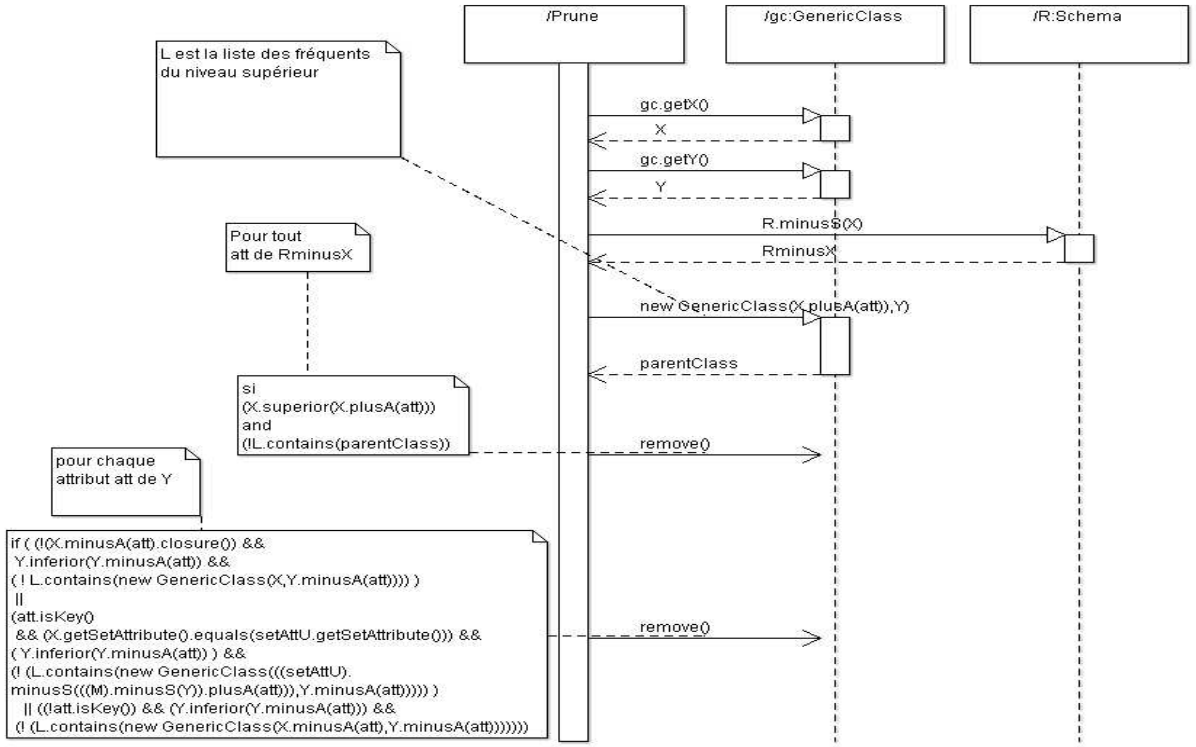


FIGURE B.3 – Diagrammme de séquences de la fonction prune()

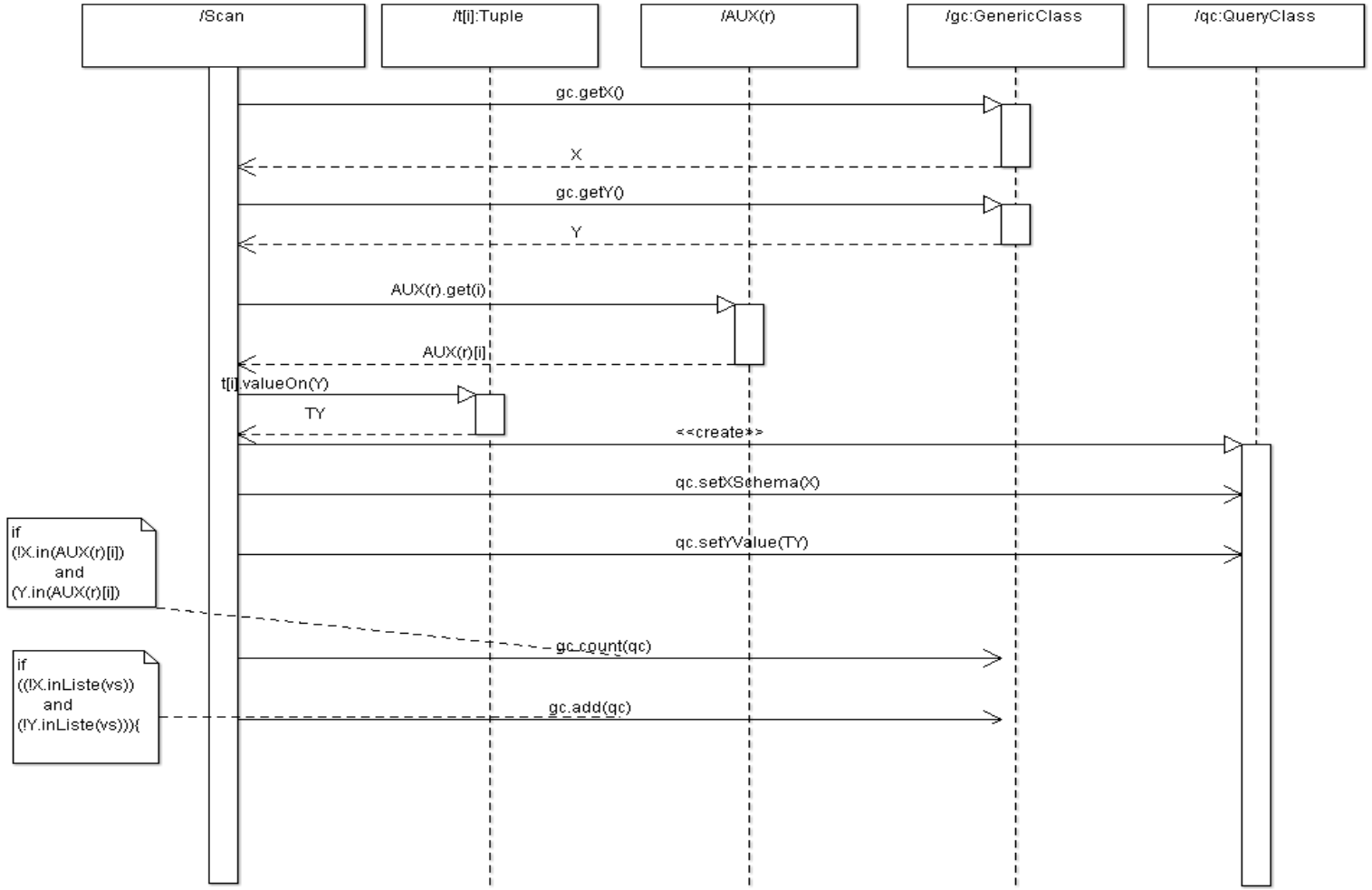


FIGURE B.4 – Diagrammme de séquences de la fonction scan()

B.3 Loader

Cette classe contient les attributs et les méthodes permettant de se connecter par jdbc à notre base de données mysql mais également , de générer à la volet la table de jointure J mais également de parcourir un à un les tuples de la base. Le tableau suivant décrit les attributs et les méthodes les plus importantes de cette classe :

Attributes	url : login : password :	String String String	
Methods	Les méthodes de cette classe correspondent aux méthodes décrites dans notre algorithme principale		
	allTuple	Parameters Description	$i : int$ permet de charger les tuples de la table i.
	compute_J	No Parameters Returns	 la table de jointure J

B.4 Aux

Cette classe contient les méthodes permettant de calculer la table AUX d'une table r telle que définie dans cette thèse : Le tableau suivant décrit les attributs et les méthodes les plus importants de cette classe :

Attributes	url : login : password : tableid :	String String String int	
Methods	Les méthodes de cette classe correspondent aux méthodes décrites dans notre algorithme principale		
	getaux	Parameters Description	tab_of_tuples : $Object[]$ OUT load :Loader i :int tab_of_tuples contient la table AUX de la table i

B.5 Tuple

Cette classe représente la notion de tuple telle que définie dans cette thèse. Le tableau suivant décrit les attributs et les méthodes les plus importantes de cette classe :

Attributes	mapAttToValues	$HashMap < Integer, String >$	
Methods	Les méthodes de cette classe correspondent aux méthodes décrites dans notre algorithme principale		
	addElement	Parameters Description	i :int s :String ajoute (i,s) à this i represente le i-eme attribut et s sa valeur
	valueOn	Parameters Returns	X : Schema this.X

B.6 Saver

Cette classe permet de sauvegarder dans un fichier les requêtes fréquentes d'une table r obtenues ainsi que leur support dans un fichier pour leur exploitation future.

Attributes	OUT_FILE_NAME_TXT	String	
Methods	saveQueries	Parameters Description	$setOfQueryClass$: $HashSet < QueryClass >$ permet de sauvegarder les requêtes fréquentes de la table t_i dans OUT_FILE_NAME_TXT.

B.7 Schema

Cette classe représente la notion de schéma telle que définie dans cette thèse. La classe Schema représente un ensemble d'attributs, Schema est représenté par un tableau de bits dans lequel chaque bit i représente un attribut A_i de U tels que si le bit i est à 1 l'attribut A_i est présent dans le schéma sinon l'attribut A_i est absent du schéma. Ainsi, la taille en octets d'un schéma est au maximum $|U|/8$ octets, ce qui permet d'utiliser le minimum d'espace mémoire possible. Le tableau suivant décrit les attributs et les méthodes les plus importantes de cette classe :

Attributes	setOfAttribute :	BitSet	
Methods	Les méthodes de cette classe correspondent aux méthodes décrites dans notre algorithme principale		
	addAttribute	Parameters Description	$i : int$ ajoute l'attribut i au schema
	closure	NO Parameters Returns	 true si le schema est clos false sinon
	contains	i : int Returns	 true si *this* contient l'attribut numero false sinon
	equals	X : Schema Returns	 true si X égale à this false sinon
	hashCode	No Parameters Returns Description	 le hashcode du schéma Cette fonction est utilisée pour faciliter les recherches.
	inclus	Parameters X :Schema Returns	 true si X inclus dans this false sinon
	inferior	Parameters X :Schema Returns	 true si $X \in this\downarrow$ false sinon
	inter	Parameters X : Schema Returns Description	 $this \cap X$ Cette fonction retourne l'intersection de X et this.
	minusA	Parameters i :int Returns	 *this* moins le i-eme attribut
	minusS	Parameters X : Schema Returns	 this moins le schema X
	plusA	Parameters i :int Returns	 *this* plus le i-eme attribut
	superior	Parameters X :Schema Returns	 true si $X \in this\uparrow$ false sinon

B.8 GenericClass

Cette classe représente la notion de Generic Class telle que présentée dans cette thèse. Le tableau suivant décrit les attributs et les méthodes les plus importants de cette classe :

Attributes	xGenericClass yGenericClass	Schema Schema	
Methods	Les méthodes de cette classe correspondent aux méthodes décrites dans notre algorithme principale		
	addQueryClass	Parameters Description	$qc : QueryClass$ ajoute qc à $setOfQueryClass$
	contains	Parameters Returns	$qc : QueryClass$ true si qc est dans setOfQueryClass false sinon
	generate	No Parameters Returns Description	 $HashSet < GenericClass > C$ C :Ensemble des classes génériques filles.
	hashCode	No Parameters Returns Description	 le hashcode de la classe générique Cette fonction est utilisée pour faciliter les recherches.
	free	No Parameters force la liberation de l'espace mémoire occupé	

B.9 QueryClass

Cette classe permet de représenter la notion de requête telle que présentée dans cette thèse. Le tableau suivant décrit les attributs et les méthodes les plus importantes de cette classe :

Attributes	support xSchema ySchema yValue	int Schema Schema Tuple	
Methods	Les méthodes de cette classe correspondent aux méthodes décrites dans notre algorithme principale		
	free	No Parameters force la liberation de l'espace mémoire occupé	

B.10 Frequent Queries

Cette classe est la classe principale de l'application.

Le tableau suivant décrit les attributs et les méthodes les plus importantes de cette classe :

Attributes	threshold load	Real Loader	 Instance de la classe Loader
Methods	Les méthodes de cette classe correspondent aux méthodes décrites dans notre algorithme principale		
	mine	Parameters Description	$int\ i$ $HashSet\langle QueryClass\rangle\ LFreq$ OUT i :numero de la table sur lequel on applique mine $LFreq$:liste des requêtes fréquentes de la table i
	generate	Parameters Returns Description	$HashSet < GenericClass > L$ IN $HashSet < GenericClass > C$ L :liste des classes fréquentes du niveau l C :liste des classes candidates du niveau $l+1$
	prune	Parameters Returns Description	$HashSet < GenericClass > C$ $HashSet < GenericClass > L$ OUT $HashSet < GenericClass > C$ C :liste des classes génériques candidates l
	scan	Parameters Returns Description	$HashSet < GenericClass > L$ $HashSet < GenericClass > C$ L :liste des classes fréquentes du niveau l C :liste des classes candidates du niveau $l+1$

Résumé

Au cours de ces dernières années, le problème de la recherche de requêtes fréquentes dans les bases de données est un problème qui a suscité de nombreuses recherches. En effet, beaucoup de motifs intéressants comme les règles d'association, des dépendances fonctionnelles exactes ou approximatives, des dépendances fonctionnelles conditionnelles exactes ou approximatives peuvent être découverts simplement, contrairement au méthodes classiques qui requièrent plusieurs transformations de la base pour extraire de tels motifs. Cependant, le problème de la recherche de requêtes fréquentes dans les bases de données relationnelles est un problème difficile car, d'une part l'espace de recherche est très grand (puisque égal à l'ensemble de toutes les requêtes pouvant être posées sur une base de données), et d'autre part, savoir si deux requêtes sont équivalentes (donc engendrant les calculs de support redondants) est un problème NP-Complet. Dans cette thèse, nous portons notre attention sur les requêtes de type Projection-Selection-Jointure (PSJ), et nous supposons que la base de données est définie selon un schéma étoile. Sous ces hypothèses, nous définissons une relation de pré-ordre ($\leq$) entre les requêtes et nous montrons que :

1. La mesure de support est anti-monotone par rapport à $\leq$, et
2. En définissant, $q \equiv q'$ si et seulement si $q \leq q'$ et $q' \leq q$, alors toutes les requêtes d'une même classe d'équivalence ont même support.

Les principales contributions de cette thèse sont, d'une part d'étudier formellement les propriétés du pré-ordre et de la relation d'équivalence ci-dessus, et d'autre part, de proposer un algorithme par niveau de type Apriori pour rechercher l'ensemble des requêtes fréquentes d'une base de données définie sur un schéma étoile. De plus, cet algorithme a été implémenté et les expérimentations que nous avons réalisées montrent que, selon notre approche, le temps de calcul des requêtes fréquentes dans une base de données définie sur un schéma étoile reste acceptable, y compris dans le cas de grandes tables de faits.

Mots-clés: base de données, fouilles de données, requêtes, motifs, algorithme par niveau, connaissances

Abstract

The problem of mining frequent queries in a database has motivated many research efforts during the last two decades. This is so because many interesting patterns, such as association rules, exact or approximative functional dependencies and exact or approximative conditional functional dependencies can be easily retrieved, which is not possible using standard techniques. However, the problem mining frequent queries in a relational database is not easy because, on the one hand, the size of the search space is huge (because encompassing all possible queries that can be addressed to a given database), and

on the other hand, testing whether two queries are equivalent (which entails redundant support computations) is NP-Complete. In this thesis, we focus on Projection-Selection-Join (PSJ) queries, assuming that the database is defined over a star schema. In this setting, we define a pre-ordering ($q \leq q'$) between queries and we prove the following basic properties :

1. The support measure is anti-monotonic with respect to $\leq$, and
2. Defining $q \equiv q'$ if and only if $q \leq q'$ and $q' \leq q$, all equivalent queries have the same support.

The main contributions of the thesis are, on the one hand to formally study properties of the pre-ordering and the equivalence relation mentioned above, and on the other hand, to prose a level-wise, Apriori like algorithm for the computation of all frequent queries in a relational database defined over a star schema. Moreover, this algorithm has been implemented and the reported experiments show that, in our approach, runtime is acceptable, even in the case of large fact tables.

Keywords: database, datamining, query, pattern, level-wise algorithm, knowledge