LOGIQUE ET

BASES DE DONNÉES

1. INTRODUCTION

Historiquement, les chercheurs ont tout d'abord essayé de modéliser les langages d'interrogation de bases de données par la logique. Ainsi sont nés les calculs relationnels, véritables langages d'interrogation fondés sur la logique du premier ordre. Ce n'est qu'à partir de la fin des années 70 que l'on a cherché à appréhender globalement données et requêtes par la logique. Cela a permis de montrer qu'un SGBD était un démonstrateur de théorèmes très particulier, raisonnant sur des faits et des règles, générant toutes les preuves d'un théorème en réponse à une question représentant ce théorème.

Les travaux sur l'introduction dans les SGBD de raisonnements généraux incluant non seulement des faits, mais aussi des règles déductives exprimées en logique du premier ordre, ont débuté principalement au CERT à Toulouse à la fin de la décennie 1970-1980 [Gallaire78]. Ils se sont surtout concentrés sur la compréhension des bases de données déductives par la logique. Afin de permettre la gestion de grandes bases de connaissances (quelques milliers de règles associées à quelques millions voire milliards de faits), des problèmes de recherche complexes ont dû être résolus. Ces problèmes sont à la croisée des chemins des bases de données et de l'intelligence artificielle et présentent des aspects à la fois théoriques et pratiques. D'un point de vue théorique, les approches par la logique permettent une meilleure compréhension des langages des SGBD, des mécanismes de raisonnements sous-jacents et des techniques d'optimisation. D'un point de vue pratique, le développement de SGBD basés sur la logique supportant la déduction a conduit à des prototypes opérationnels, mais très rarement à des produits industriels.

Ce chapitre se propose tout d'abord d'introduire les notions de base de logique nécessaires à la bonne compréhension des bases de données. Nous définissons ce qu'est une base de données logique vue comme une interprétation d'un langage logique. Il s'agit simplement d'un ensemble de faits listés dans des tables. Historiquement, cette vision est une compréhension des bases de données relationnelles proposées par Gallaire, Minker et Nicolas à la fin des années 70 [Gallaire81]. C'est une vision élémentaire appelée théorie du modèle (ou de l'interprétation). Pour beaucoup, le terme BD logique inclut les facilités déductives que nous étudierons dans le chapitre sur les BD déductives.

Dans ce chapitre, nous introduisons aussi les langages logiques d'interrogation de bases de données que sont les calculs de domaines et de tuples. Ils sont une formalisation des langages des bases de données relationnelles que nous étudierons dans la partie suivante. Pour bien les comprendre, il est bon d'avoir quelques notions sur les BD relationnelles, mais ce n'est sans doute pas indispensable.

Ce chapitre comporte cinq sections. Après cette introduction, la section 2 introduit la logique du premier ordre et les manipulations de formules. La section 3 définit ce qu'est une BD logique non déductive. La section 4 est consacrée au calcul de domaine et à son application pratique, le langage QBE. La section 5 traite de la variante calcul de tuple, qui était à la base du langage QUEL, somme toute assez proche de SQL. La section 6 introduit les techniques de raisonnement et de preuve de théorème. Il est nécessaire de connaître un peu ces techniques pour aborder ultérieurement les bases de données déductives.

2. LA LOGIQUE DU PREMIER ORDRE

Dans cette partie, nous rappelons les principes de la logique du premier ordre et nous introduisons la méthode de réécriture sous forme de clauses. La logique permet une modélisation simple des bases de données, l'introduction de langages de requêtes formels et le support de la déduction.

2.1 Syntaxe de la logique du premier ordre

Rappelons que la logique du premier ordre, aussi appelée **calcul de prédicats**, est un langage formel utilisé pour représenter des relations entre objets et pour déduire de nouvelles relations à partir de relations connues comme vraies. Elle peut être vue comme un formalisme utilisé pour traduire des phrases et déduire de nouvelles phrases à partir de phrases connues.

La logique du premier ordre repose sur un alphabet qui utilise les symboles suivants :

1. Des variables généralement notées x, y, z …

2. Des constantes généralement notées a, b, c …

3. Des prédicats généralement notées P, Q, R …, chaque prédicat pouvant recevoir un nombre fixe d'arguments (n pour un prédicat n-aire).

4. Les connecteurs logiques $\wedge$, $\vee$, $\Rightarrow$, $\neg$.

5. Des fonctions généralement dénotées f, g, h,…, chaque fonction pouvant recevoir un nombre fixe d'arguments (n pour une fonction n-aire).

6. Les quantificateurs $\forall$ et $\exists$.

Des règles syntaxiques simples permettent de construire des formules. Un **terme** est défini récursivement comme une variable ou une constante ou le résultat de l'application d'une fonction à un terme. Par exemple, x, a, f(x) et f(f(x)) sont des termes. Une **formule** est une phrase bien construite du langage. Une formule est définie comme suit :

1. Si P est un prédicat à n arguments (n places) et t1, t2,…tn des termes, alors P(t1,t2,…tn) est une **formule atomique**.

2. Si F1 et F2 sont des formules, alors F1 $\wedge$ F2, F1 $\vee$ F2, F1 $\Rightarrow$ F2 et $\neg$F1 sont des formules.

3. Si F est une formule et x une variable non quantifiée (on dit aussi libre) dans F, alors $\forall$x F et $\exists$x F sont des formules.

Nous résumons ci-dessous les concepts essentiels introduits jusque-là sous forme de notions.

> **Notion V.1 : Terme (*Term*)**
>
> Constante, variable ou fonction appliquée à un terme.

> **Notion V.2 : Formule atomique (*Atomic Formula*)**
>
> Résultat de l'application d'un prédicat à n-places à n termes, de la forme P(t1,t2,…tn).

> **Notion V.3 : Formule (*Formula*)**
>
> Suite de formules atomiques ou de négation de formules atomiques séparées par des connecteurs logiques `et`, `ou`, `implique`, avec d'éventuelles quantifications des variables.

Pour indiquer les priorités entre connecteurs logiques, il est possible d'utiliser les parenthèses : si F est une formule valide, (F) en est aussi une. Afin d'éviter les parenthèses dans certains cas simples, nous supposerons une priorité des opérateurs logiques dans l'ordre descendant $\neg$, $\vee$, $\wedge$, et $\Rightarrow$. Voici quelques exemples de formules formelles :

P(a,x) $\wedge$ Q(x,y),

$\neg Q(x,y)$,

$\forall x \, \exists y \, (Q(x,y) \vee P(a,x))$,

$\forall x \, (R(x) \wedge Q(x,a) \Rightarrow P(b,f(x)))$.

Afin de donner des exemples plus lisibles de formules, nous choisirons un vocabulaire moins formel, les prédicats et fonctions étant des noms ou des verbes du langage courant et les constantes des chaînes de caractères désignant par exemple des services ou des employés. Voici quelques exemples de formules proches du langage naturel :

$$\text{SERVICE(informatique,pierre)} \vee \text{EMPLOYE(informatique,marie)}$$

$$\forall x \, ((\text{DIRIGE(pierre,x)} \vee \neg \text{EMPLOYE(informatique,x)} \Rightarrow \text{EMPLOYE(finance,x)})$$

$$\forall x \, \exists y \, (\, (\, \text{DIRIGE}(x,y) \vee \text{DIRIGE}(y,x)\,) \Rightarrow \text{MEMESERVICE}(x,y)\,)$$

2.2 Sémantique de la logique du premier ordre

Une formule peut être interprétée comme une phrase sur un ensemble d'objets : il est possible de lui donner une signification vis-à-vis de cet ensemble d'objets. Pour ce faire, il faut assigner un objet spécifique à chaque constante. Par exemple, on assigne un objet à la constante `a`, le service Informatique de l'entreprise considérée à la constante `informatique`, l'employée Marie à la constante `marie`, etc. L'ensemble d'objets considérés est appelé le **domaine de discours;** il est noté D. Chaque fonction à n arguments est interprétée comme une fonction de D^n dans D. Un prédicat représente une relation particulière entre les objets de D, qui peut être vraie ou fausse. Définir cette relation revient à définir les tuples d'objets qui satisfont le prédicat. L'ensemble des tuples satisfaisants le prédicat constitue son extension.

> **Notion V.4 : Domaine de discours (*Domain of Discourse*)**
>
> Ensemble d'objets sur lequel une formule logique prend valeur par interprétation des constantes comme des objets particuliers, des variables comme des objets quelconques, des prédicats comme des relations entre objets et des fonctions comme des fonctions particulières entre objets.

Par exemple, la formule :

$$\forall x \, \exists y \, (\, (\, \text{DIRIGE}(x,y) \vee \text{DIRIGE}(y,x)\,) \Rightarrow \text{MEMESERVICE}(x,y)\,)$$

peut être interprétée sur l'ensemble des personnes {Pierre, Paul, Marie}, qui constitue alors le domaine de discours. DIRIGE peut être interprété comme la relation binaire « est chef de » ; MEMESERVICE peut être interprété comme la relation binaire « travaille dans le même service ». La formule est vraie si Pierre, Paul et Marie travaillent dans le même service.

Un univers de discours infini peut être retenu pour interpréter la formule $\forall x$ (x < SUCC(x)). En effet, cette formule peut être interprétée sur l'ensemble des entiers positifs {1,2,3,…} qui est infini. < est alors la relation « est inférieur à » et SUCC la fonction qui à tout entier associe son successeur. Il est clair que cette formule est vraie sur les entiers.

Ainsi, étant donné une interprétation d'une formule sur un domaine de discours, il est possible d'associer une valeur de vérité à cette formule. Pour éviter les ambiguïtés (les formules pouvant avoir plusieurs valeurs de vérité), nous considérons seulement les formules dans lesquelles toutes les variables sont quantifiées, appelées **formules fermées**. Toute formule fermée F a une unique valeur de vérité pour une interprétation donnée sur un domaine de discours D. Cette valeur notée V(F) se calcule ainsi :

$V(F1 \vee F2) = V(F1) \vee V(F2)$

$V(F1 \wedge F2) = V(F1) \wedge V(F2)$

$V(\neg F1) = \neg V(F1)$

$V(\forall x\ F) = V(F\ x=a) \wedge V(F\ x=b) \wedge \ldots$ où a, b, … sont les constantes de D.

$V(\exists x\ F) = V(F\ x=a) \vee V(F\ x=b) \vee \ldots$ où a, b, … sont les constantes de D.

V(P(a,b…)) = Vrai si [a,b,…] satisfait la relation P et Faux sinon.

Un **modèle** d'une formule logique est une interprétation sur un domaine de discours qui rend la formule vraie. Par exemple, les entiers sont un modèle pour la formule $\forall x$ (x < SUCC(x)) avec l'interprétation indiquée ci-dessus. En effet, en appliquant les règles ci-dessus, on calcule :

$V(\forall x\ (x < SUCC(x))) = V(1 < 2) \wedge V(2<3) \wedge V(3<4) \wedge \ldots$ = Vrai.

2.3 Forme clausale des formules fermées

Toute formule fermée, c'est-à-dire à variables quantifiées, peut se simplifier et s'écrire sous une forme canonique sans quantificateurs, appelée forme clausale. La forme clausale nécessite d'écrire la formule comme un ensemble de **clauses.**

> **Notion V.5 : Clause (*Clause*)**
>
> Formule de la forme $P1 \wedge P2 \wedge \ldots \wedge Pn \Rightarrow Q1 \vee Q2 \vee \ldots \vee Qm$, ou les Pi et les Qj sont des littéraux positifs (c'est-à-dire des prédicats atomiques sans négation devant).

Une clause ayant un seul littéral situé après l'implication (on dit en tête de clause), donc un seul Qi, est dite **clause de Horn**.

Notion V.6 : Clause de Horn (*Horn Clause*)

Clause de la forme P1$\wedge$ P2 $\wedge$ … $\wedge$ Pn $\Rightarrow$ Q1.

Par exemple :

ENTIER(x) $\wedge$ (x > 0) $\Rightarrow$ (x < SUCC(x)) $\vee$ (x > SUCC(x))

DIRIGE(x,y) $\wedge$ MEMESERVICE(x,y) $\Rightarrow$ AIME(x,y) $\vee$ DETESTE(x,y)

sont des clauses.

ENTIER(x) $\wedge$ (x > 0) $\Rightarrow$ (x < SUCC(x))

DIRIGE(x,y) $\wedge$ MEMESERVICE(x,y) $\Rightarrow$ AIME(x,y)

sont des clauses de Horn.

La transformation d'une formule fermée en clauses s'effectue par des transformations successives que nous décrivons brièvement ci-dessous. Nous illustrons les transformations avec la formule :

$\forall x \exists y$ ((DIRIGE(x,y) $\vee$ DIRIGE(y,x)) $\Rightarrow$ MEMESERVICE(x,y)).

1. **Élimination des implications.** Ceci s'effectue simplement en remplaçant toute expression de la forme A $\Rightarrow$ B par $\neg$A $\vee$ B. En effet, ces expressions sont équivalentes du point de vue logique. La formule choisie en exemple devient :

   $\forall x \exists y$ ($\neg$(DIRIGE(x,y) $\vee$ DIRIGE(y,x)) $\vee$ MEMESERVICE(x,y)).

2. **Réduction de la portée des négations.** Le but est de faire que les négations s'appliquent directement à des prédicats atomiques. Pour cela, on utilise de manière répétée les transformations :

   $\neg$(A $\vee$ B) devient $\neg$A $\wedge$ $\neg$B;

   $\neg$(A $\wedge$ B) devient $\neg$A $\vee$ $\neg$B;

   $\neg(\forall x$A) devient $\exists x\neg$A;

   $\neg(\exists x$A) devient $\forall x\neg$A;

   $\neg(\neg$A) devient A.

   L'exemple devient :

   $\forall x \exists y$ (($\neg$DIRIGE(x,y) $\wedge$ $\neg$DIRIGE(y,x)) $\vee$ MEMESERVICE(x,y)).

3. **Mise en forme prenex.** Il s'agit simplement de mettre les quantificateurs avec la variable quantifiée en-tête de formule. Cette étape peut nécessiter de renommer les variables afin d'éviter les confusions de quantification. Notre exemple reste ici inchangé.

4. **Élimination des quantificateurs.** Afin de simplifier encore, les variables quantifiées existentiellement sont remplacées par une constante paramètre dite **constante de Skolem**. En effet, s'il existe x satisfaisant une formule, il est possible de choisir une constante s désignant cette valeur de x. Attention, si une variable quantifiée par « quel que soit » apparaît avant la variable quantifiée par « il existe », la constante choisie dépend de la première variable. Par exemple, dans le cas $\forall x \exists y$ (pour tout x, il existe y, mais le y dépend de x), on remplacera donc y par s(x), c'est-à-dire une fonction de Skolem qui matérialise la constante y dépendant de x. Ainsi, il est possible d'éliminer les variables quantifiées par « il existe ». Après cette transformation, toute variable restante est quantifiée par « quel que soit ». On peut donc oublier d'écrire les quantificateurs (c'est implicitement $\forall$). Notre exemple devient alors :

   $((\neg DIRIGE(x,s(x)) \wedge \neg DIRIGE(s(x),x)) \vee MEMESERVICE(x,s(x)))$.

5. **Mise en forme normale conjonctive.** La formule restante est une combinaison par des « ou » et des « et » de littéraux positifs ou négatifs (prédicats atomiques précédés ou non d'une négation). Elle peut être écrite comme une conjonction ($\wedge$) de disjonction ($\vee$) en distribuant les « et » par rapport aux « ou » à l'aide de la règle :

   $A \vee (B \wedge C)$ devient $(A \vee B) \wedge (A \vee C)$.

   L'exemple devient ainsi :

   $(\neg DIRIGE(x,s(x)) \vee MEMESERVICE(x,s(x))) \wedge (\neg DIRIGE(s(x),x)) \vee MEMESERVICE(x,s(x)))$.

6. **Écriture des clauses.** Finalement, il est simple d'écrire chaque clause sur une ligne en remplaçant les « et » par des changements de lignes. De plus, profitant du fait que $\neg A \vee B$ s' écrit $A \Rightarrow B$, on peut écrire tous les prédicats négatifs d'abord en éliminant la négation et en les connectant par $\wedge$, puis tous les prédicats positifs connectés par $\vee$. On obtient bien ainsi une suite de clauses, définies comme ci-dessus. Avec l'exemple, on obtient deux clauses (d'ailleurs de Horn):

   $DIRIGE(x,s(x)) \Rightarrow MEMESERVICE(x,s(x))$

   $DIRIGE(s(x),x)) \Rightarrow MEMESERVICE(x,s(x)))$.

3. LES BASE DE DONNÈES LOGIQUE

La notion de base de données logique a été introduite en particulier à Toulouse à la fin des années 70 [Gallaire84]. Nous définissons ici ce qu'est une base de données logique non déductive. Il s'agit d'une première approche simple des bases de données par la logique. Nous verrons plus loin que cette approche peut être étendue avec la déduction.

3.1 La représentation des faits

Une base de données peut être définie comme **l'interprétation d'un langage logique** du premier ordre L. En pratique, définir l'interprétation consiste à définir les prédicats vrais. Le langage permet d'exprimer les requêtes, comme nous le verrons ci-dessous. Une contrainte d'intégrité peut être vue comme une requête toujours vraie, donc exprimée avec le langage.

> **Notion V.7 : Base de données logique (*Logic database*)**
>
> Ensemble de faits constituant l'interprétation d'un langage du premier ordre avec lequel il est possible d'exprimer des questions et des contraintes d'intégrité sur les faits.

En première approche, le langage logique L ne comporte pas de fonctions. Plus particulièrement, L se compose :

1. de prédicats à n arguments, chaque argument correspondant à un type de données élémentaire ;

2. de constantes, une pour chaque valeur possible des type de données élémentaires.

Comme dans toute interprétation d'un langage logique, un prédicat représente une relation particulière entre les objets du domaine de discours, qui peut être vraie ou fausse. Ici les objets du domaine de discours sont donc les valeurs possibles de la base. Définir cette relation revient à définir les articles ou n-uplets qui satisfont le prédicat. L'ensemble des n-uplets satisfaisant le prédicat constitue son **extension**. Cette extension peut être assimilée à un fichier dans une base en réseau ou à une table relationnelle, comme nous le verrons plus loin. Pour rester cohérent avec les bases de données relationnelles qui peuvent être perçues comme une implémentation simple des bases de données logiques, nous appellerons l'extension d'un prédicat une **table**. Chaque colonne correspond à un argument et est aussi appelée **attribut** dans le contexte relationnel. Comme déjà dit, une base de données logique pourra être complétée par des règles de déduction : nous parlerons alors de **base de données déductive** (voir partie 4 de cet ouvrage).

La figure V.1 illustre une base de données logique. En termes de tables, cette base correspond à celles représentées figure V.2. Elle est composée de trois prédicats définis comme suit :

- PRODUIT avec les arguments Numéro de produit (NPRO), nom de produit (NPRO), quantité en stock (QTES), et couleur (COULEUR) ;

- VENTE avec les arguments numéro de vente (NVEN), nom du client (NOMC), numéro du produit vendu (NPRV), quantité vendue (QTEV) et date (DATE) ;

- ACHAT avec les arguments numéro d'achat (NACH), date de l'achat (DATE), numéro du produit acheté (NPRA), quantité achetée (QTEA) et nom du fournisseur (NOMF).

PRODUIT (100, BILLE, 100 , VERTE)
PRODUIT (200, POUPEE, 50, ROUGE)
PRODUIT (300, VOITURE, 70, JAUNE)
PRODUIT (400, CARTE, 350, BLEU)
VENTE (1, DUPONT, 100, 30, 08-03-1999)
VENTE (2, MARTIN, 200, 10, 07-01-1999)
VENTE (3, CHARLES, 100, 50, 01-01-2000)
VENTE (4, CHARLES, 300, 50, 01-01-2000)
ACHAT (1, 01-03-1999, 100, 70, FOURNIER)
ACHAT (2, 01-03-1999, 200, 100, FOURNIER)
ACHAT (3, 01-09-1999, 100, 50, DUBOIS)
ACHAT (4, 01-09-1999, 300, 50, DUBOIS)

Figure V.1 — Faits constituant une base de données logique

PRODUIT

NPRO	NOMP	QTES	COULEUR
100	Bille	100	Verte
200	Poupée	50	Rouge
300	Voiture	70	Jaune
400	Carte	350	Bleu

VENTE

NVEN	NOMC	NPRV	QTEV	DATE
1	Dupont	100	30	08-03-1999
2	Martin	200	10	07-01-1999
3	Charles	100	50	01-01-2000
4	Charles	300	50	01-01-2000

ACHAT

NACH	DATE	NPRA	QTEA	NOMF
1	01-03-1999	100	70	Fournier
2	01-03-1999	200	100	Fournier
3	01-09-1999	100	50	Dubois
4	01-09-1999	300	50	Dubois

Figure V.2 — Tables représentant la base de données logique

Comme on le voit, seul les faits positifs, c'est-à-dire ceux satisfaisant l'un des trois prédicats, sont enregistrés dans la base. Ceci constitue **l'hypothèse du monde fermé**. Les faits non enregistrés dans une extension de prédicat sont supposés faux. Il est vrai que si vous interrogez la base de données que gère votre banque et que vous ne voyez pas de chèque de 100.000 euros crédités ce jour, c'est que le fait que cette somme ait été créditée sur votre compte est faux ! Des chercheurs ont essayé de lever cette hypothèse : on tombe alors dans l'hypothèse du monde ouvert, ou un fait non enregistré peut être inconnu [Reiter78].

3.2 Questions et contraintes d'intégrité

Les questions et les contraintes d'intégrité sur la base peuvent alors être exprimées comme des formules dans le langage logique. Cependant, celui-ci doit être étendu pour inclure les opérateurs de comparaison arithmétique {=, <, ≤, >, ≥, ≠ } comme des prédicats particuliers, dont la valeur de vérité est calculée par les opérations usuelles des calculateurs.

La réponse à une question F(x1, …, xn) — où x1, …xn sont des variables libres dans la formule F — est alors l'ensemble des n-uplets <e1, …,ep> tels que F(e1, …ep) est vraie. Certaines formules doivent être toujours vraies sur l'interprétation que constitue la base : ce sont alors des contraintes d'intégrité. Cette vue logique des bases de données a donné naissance à deux calculs permettant d'exprimer des questions et des contraintes d'intégrité sur les bases : le calcul de domaine et le calcul de tuple. Dans le premier, les objets de l'interprétation logique sont les valeurs atomiques de données ; dans le second ce sont les faits composites correspondant aux n-uplets, encore appelés tuples. Nous étudions ces deux calculs ci-dessous.

4. LE CALCUL DE DOMAINES

Les calculs relationnels de domaine et de tuples [Codd72] permettent d'exprimer des requêtes à l'aide de formules du premier ordre sur une base de données logique, ou sa représentation tabulaire qui est une base de données relationnelles. Ils ont été utilisés pour formaliser les langages d'interrogation pour les bases de données relationnelles. Ci-dessous, nous étudions tout d'abord le calcul de domaine et sa représentation bi-dimensionnelle QBE [Zloof77], puis le calcul de

tuples. Le langage QUEL introduit au chapitre II peut être perçu comme un paraphrasage en anglais du calcul de tuples.

4.1 Principes de base

Le calcul relationnel de domaines [Codd72] se déduit donc de la logique du premier ordre. Les données sont les constantes. Les prédicats utilisés sont :

1. les **prédicats extensionnels** contenant les enregistrements de la base ; chaque enregistrement est un tuple typé selon un prédicat extensionnel ; les prédicats étant n-aire, une variable ou une constante est utilisée comme argument de prédicat ; les variables sont alors implicitement typées selon le type de l'argument pour lequel elles apparaissent ;

2. les **prédicats de comparaison** entre une variable et une constante, ou entre deux variables, construits à l'aide des opérateurs {=, ≤, <, ≥, >, ≠}.

Une question en calcul relationnel de domaine s'écrit donc sous la forme suivante :

$$\{x, y, \ldots \mid F(x,y, \ldots)\}$$

F est une formule logique composée à partir de prédicats extensionnels et de comparaison ; les variables résultats x, y, …, sont des variables libres dans F.

La notion suivante résume la définition du calcul de domaine.

> **Notion V.8 : Calcul relationnel de domaines (*Domain relational calculus*)**
>
> Langage d'interrogation de données formel permettant d'exprimer des questions à partir de formules bien formées dont chaque variable est interprétée comme variant sur le domaine d'un argument d'un prédicat.

La BNF du calcul relationnel de domaine est définie figure V.3. Pour simplifier, les formules sont écrites en forme prenex (quantificateurs devant). En pratique, une simplification d'écriture permet de regrouper des prédicats de restriction du type (x = a) et des prédicats de définition de variable du type P (x, …) en écrivant P (a, …). Toute variable apparaissant dans le résultat doit être définie dans un prédicat extensionnel et doit rester libre dans la formule spécifiant la condition. Une variable non utilisée ni en résultat ni dans un critère de comparaison peut être remplacée par la variable muette positionnelle notée "-".

<question> : := “{” (<résultat>) “|” <formule> “}”

<résultat> : := <variable> | <variable>, <résultat>

<formule> : := <quantification> <formule libre> | <formule libre>

<quantification> : := ∀ <variable> | ∃ <variable>

| <quantification> <quantification>

<formule libre> : := <formule atomique>

| <formule libre> ∧ <formule atomique>

| <formule libre> ∨ <formule atomique>

| <formule libre> ⇒ <formule libre>

| ¬ <formule libre>

| (<formule libre>)

<formule atomique> : := <prédicat extensionnel> (<arguments>)

| <terme> <comparateur> <terme>

<arguments> : := <terme> | <terme>, <terme>

<terme> : := <variable> | <constante>

<comparateur> : := = | < | ≤ | > | ≥ | ≠

Figure V.3 — BNF du calcul relationnel de domaines

4.2 Quelques exemples de calcul de domaine

Nous illustrons le calcul de domaine sur la base logique introduite ci-dessus décrivant des achats et des ventes de produits stockés dans un magasin. Le schéma de la base est le suivant :

PRODUIT (NPRO, NOMP, QTES, COULEUR)

VENTE (NVEN, NOMC, NPRV, QTEV, DATE)

ACHAT (NACH, DATE, NPRA, QTEA, NOMF)

Le prédicat extensionnel PRODUIT se compose des attributs numéro de produit (NPRO), nom du produit (NOMP), quantité en stock (QTES) et couleur (COULEUR). Le prédicat VENTE décrit les ventes de produits effectuées et se compose des attributs numéro de vente (NVEN), nom du client (NOMC), numéro du produit vendu (NPRV), quantité vendue (QTEV) et date de la vente (DATE). Le prédicat ACHAT définit les achats effectués aux fournisseurs. Il contient les attributs numéro d'achat (NACH), date d'achat (DATE), numéro du produit acheté (NPRA), quantité achetée (QTEA) et nom du fournisseur (NOMF). Pour simplifier la lecture, nous utilisons des variables rappelant le nom du domaine. Notez que les quantificateurs existentiels dans les formules peuvent être omis, car ils sont implicites si la variable est libre et ne figure pas en résultat.

(Q1) Donner la liste des noms et des couleurs de tous les produits :

$\{(p,c) \mid \text{PRODUIT}(-,p,-,c)\}$

(Q2) Donner les noms et les quantités en stock des produits de couleur rouge :

$\{(p,q) \mid \text{PRODUIT}(-, p, q, \text{"Rouge"})\}$

(Q3) Donner, pour chaque produit en stock, le nom du fournisseur associé :

$\{(p,f) \mid \exists n\ (\text{PRODUIT}(n, p, -, -) \wedge \text{ACHAT}(-, -, n, -, f))\}$

(Q4) Donner, pour chaque produit en stock en quantité supérieure à 100 et de couleur rouge, les triplets nom de fournisseurs ayant vendu ce type de produit et nom de client ayant acheté ce type de produit et nom du produit :

$\{(f, c, p) \mid \exists n\ \exists q\ (\text{PRODUIT}(n, p, q, \text{"Rouge"}) \wedge \text{ACHAT}(-, -, n, -, f)$

$\wedge\ \text{VENTE}(-, c, n, -, -) \wedge (q > 100))\}$

(Q5) Donner les noms des clients ayant acheté au moins un produit de couleur verte :

$\{(c) \mid \exists n\ (\text{VENTE}(-, c, n, -, -) \wedge \text{PRODUIT}(n, -, -, \text{"Verte"}))\}$

(Q6) Donner les noms des clients ayant acheté tous les produits stockés :

$\{(c) \mid \forall p\ (\text{PRODUIT}(p, -, -, -) \Rightarrow \text{VENTE}(-, c, p, -, -))\}$

(Q7) Donner les noms des produits fournis par tous les fournisseurs et achetés par au moins un client :

$\{(p) \mid \forall f\ (\text{ACHAT}(-,-, -, -, f) \Rightarrow \text{ACHAT}(-, -, p, -, f))$

$\wedge\ \exists c\ \text{VENTE}(-, c, p, -, -)\ \}$

4.3 Le langage QBE

Le langage Query-By-Exemple (QBE) [Zloof77] est conçu pour être utilisé à partir d'un terminal. Il a été développé par M. ZLOFF à IBM Yorkton Heights. Il est commercialisé par IBM au-dessus de DB2 depuis 1983 et plus récemment par de nombreux autres constructeurs avec des présentation variées, par exemple par Microsoft dans ACCESS. Ce langage peut être considéré comme une mise en œuvre visuelle (basée sur des tableaux affichés) du calcul relationnel de domaine.

L'idée de base du langage est de faire formuler une question à l'utilisateur à partir d'un exemple d'une réponse possible à la question. Pour cela, l'utilisateur provoque tout d'abord l'affichage du schéma des tables nécessaires à la question qu'il désire poser (sous forme de squelettes de tables) par frappe du nom des tables correspondantes. Ainsi, des tables vides (avec seulement les en-têtes de colonnes et le nom de la relation associée) apparaissent sur l'écran. Elles correspondent bien sûr aux définitions des prédicats extensionnels de la base

logique. Par exemple, il est possible d'afficher le schéma des tables PRODUIT, VENTE et ACHAT comme représenté figure V.4.

<voir livre Maîtriser les BD Fig VIII.13, p. 320>

Figure V.4 — Schémas de tables affichés par QBE

Comme indiqué ci-dessus, QBE peut être vu comme une implantation à deux dimensions du calcul relationnel de domaines. Pour cela, les règles suivantes sont utilisées :

1. Les résultats (attributs à projeter en relationnel) sont définis en frappant "P." (PRINT) dans la colonne associée. Il est possible d'imprimer tous les attributs d'une table en frappant "P." dans la colonne contenant le nom de la table.

2. Les constantes sont tapées directement dans la colonne de l'attribut associé, précédées de l'opérateur de comparaison les reliant à l'attribut si ce n'est pas = (c'est-à-dire $<, \leq, >, \geq, \neq$). Dans le cas où un critère complexe est nécessaire avec des conjonctions (and) et disjonctions (or), il est possible d'ouvrir une boîte condition et de taper le critère dans cette boîte (par exemple A < 10 AND A > 5).

3. Les variables domaines sont désignées par des valeurs exemples soulignées tapées dans la colonne les spécifiant ; la valeur exemple soulignée est en fait le nom de la variable domaine ; par suite, QBE associe deux valeurs soulignées identiques comme définissant une même variable.

4. Le quantificateur "quel-que-soit" est appliqué à une variable en tapant "ALL." devant son nom (l'exemple souligné) alors que toute variable non imprimée non précédée de P. est implicitement quantifiée par "il-existe".

5. Le connecteur ou (or) est exprimé en utilisant deux lignes (deux exemples) comme si l'on avait en fait deux questions, l'une avec la partie gauche et l'autre avec la partie droite de la qualification (après mise en forme normale disjonctive).

A l'aide de ces règles simples, il est possible d'exprimer toute question du calcul relationnel de domaines. Nous avons formulé (voir figure V.5) les questions introduites ci-dessus (Q1 à Q7) sur la base des produits. Remarquez l'aspect naturel de ce langage par l'exemple qui peut être simplement paraphrasé.

<voir livre Maîtriser les BD Fig VIII.14, p. 321-322-323>

Figure V.5 — Exemples de questions QBE

QBE offre quelques possibilités supplémentaires par rapport au calcul de domaines. En particulier, il est possible d'enlever l'élimination automatique des doubles lors des projections finales en spécifiant le mot clé ALL devant une variable résultat à imprimer. Par exemple, l'édition de tous les noms des clients à qui l'on a vendu des produits (sans élimination des doubles) sera effectuée en réponse à la question représentée figure V.6.

<voir livre Maîtriser les BD Fig VIII.15, p. 323>

Figure V.6 — Non-élimination des doubles en QBE

Il est également possible d'obtenir des résultats triés par ordre croissant (mot clé AO.) ou décroissant (mot clé DO.). Par exemple, l'édition des noms de produits en stock en quantité supérieure à 100 par ordre alphabétique descendant sera obtenue par exécution de la question représentée figure V.7.

<voir livre Maîtriser les BD Fig VIII.16, p. 323>

Figure V.7 — Exemple de question avec résultats triés

De plus, QBE offre des facilités pour accomplir les opérations de type fermeture transitive de graphe. Le calcul de la fermeture transitive est impossible avec les langages basés sur le calcul de prédicats. Soit par exemple la relation représentée figure V.8. Tout composant peut aussi être un composé. Si l'on veut rechercher les composants de voiture au deuxième niveau, il est possible avec QBE d'utiliser la question représentée figure V.9.

<voir livre Maîtriser les BD Fig VIII.17, p. 324>

Figure V.8 — Table composé-composant

<voir livre Maîtriser les BD Fig VIII.18, p. 324>

Figure V.9 — Recherche des composants de second niveau

Pour rechercher les composants de niveau n à partir d'un composant (ou les composés de niveau n à partir d'un composé), il faut une question à n lignes. Ceci peut être fastidieux. Afin de simplifier, QBE autorise le mot clé L entre parenthèses précédé d'un nombre de niveaux (par exemple (2L)). Ainsi, la question précédente pourra aussi être formulée comme sur la figure V.10.

<voir livre Maîtriser les BD Fig VIII.19, p. 325>

Figure V.10 — Autre manière de rechercher les composants de second niveau

La fermeture transitive du composé VOITURE consiste à rechercher les composants de tout niveau. Ceci est effectué en fixant un niveau variable, comme représenté figure V.11. Une telle question n'est pas exprimable à l'aide du calcul relationnel de domaines, mais nécessite la récursion que nous étudierons dans le cadre des BD déductives. Il est aussi possible d'obtenir les composants de dernier niveau à l'aide du mot clé LAST.

<voir livre Maîtriser les BD Fig VIII.20, p. 325>

Figure V.11 — Fermeture transitive du composé VOITURE

Finalement, QBE dispose également des fonctions d'agrégats qui dépassent la logique du premier ordre. Les fonctions CNT (décompte), SUM (somme), AVG

(moyenne), MAX (maximum) et MIN (minimum) permettent de faire des calculs sur plusieurs valeurs de variables, celles-ci étant sélectionnées par des critères exprimés par une question. Ces fonctions peuvent être appliquées à toute variable résultat à condition qu'elle soit précédée de ALL. Si l'on désire éliminer les doubles avant application de la fonction, il faut appliquer avant l'opérateur unique (mot clé UNQ). Par exemple, si l'on veut connaître la quantité en stock des produits de nom "Vins" on écrira la question représentée figure V.12.

<voir livre Maîtriser les BD Fig VIII.21, p. 326>

Figure V.12 — Utilisation de la fonction SUM

QBE permet aussi la mise à jour des prédicats extensionnels, c'est-à-dire des tables. Il est possible :

- d'insérer des n-uplets dans une relation (opérateur I. en première colonne) ; la figure V.13 illustre l'insertion du produit de clé 200 ;

- de modifier des attributs de n-uplet (opérateur U. en première colonne) ; la figure V.14 illustre la mise à jour des quantités en stock pour les produits de couleur rouge ;

- de supprimer les n-uplets dans une relation obéissant à un critère de sélection donné (opérateur "D." en première colonne) ; la figure V.15 illustre la suppression des produits rouges de la relation produit.

<voir livre Maîtriser les BD Fig VIII.22, p. 326>

Figure V.13 — Insertion du tuple de clé 200

<voir livre Maîtriser les BD Fig VIII.23, p. 327>

Figure V.14 — Mise à jour des quantités en stock des produits rouges

<voir livre Maîtriser les BD Fig VIII.24, p. 327>

Figure V.15 — Suppression des produits rouges

QBE permet enfin une définition très dynamique de la base de données. Il est possible de créer et détruire des prédicats extensionnels (tables). De plus, il est permis d'étendre un prédicat en ajoutant des attributs. QBE est donc un langage très souple et complet, issu de la logique du premier ordre appliquée aux tables relationnelles.

5. LE CALCUL DE TUPLES

Le calcul relationnel de tuples [Codd72] se déduit de la logique du premier ordre. Comme le calcul de domaine, le calcul de tuples permet d'exprimer une question comme une formule. Cette fois, les constantes sont interprétées comme les n-uplets de la base. Les variables parcourent donc les lignes des tables. Afin de

distinguer les deux calculs, nous noterons les variables tuples par des majuscules X, Y, Z,… Il est d'ailleurs possibles de mixer calcul de domaines et calcul de tuples en utilisant à la fois des variables tuples et des variables domaines [Reiter84].

5.1 Principes du calcul de tuple

Les seuls termes considérés sont les constantes associées aux n-uplets composant les faits et les fonctions génératrices des attributs notées par le nom de l'attribut appliqué à une variable par la notation pointée (par exemple X.COULEUR). Les prédicats utilisés sont ceux correspondant aux relations extensionnelles ainsi que les prédicats de comparaison {=, <, ≤, ≥, >, ≠}. Les prédicats extensionnels permettent la définition de la portée d'une variable sur une table R par une formule atomique du type R (X), X étant donc une variable tuple et R une table.

L'élément essentiel différenciant le calcul de tuples par rapport aux calculs de domaines est bien sûr l'association aux variables de tuples des extensions de prédicats et non plus de valeurs de domaines, comme avec le calcul de domaine. Cela change légèrement la syntaxe du calcul. Nous définissons plus précisément le calcul de tuples ci-dessous. La syntaxe du calcul est définie en BNF figure V.16.

> **Notion V.9 : Calcul relationnel de tuples (*Tuple relational calculus*)**
>
> Langage d'interrogation de données formel permettant d'exprimer des questions à partir de formules bien formées dont les variables sont interprétées comme variant sur les tuples d'une table.

<question> : := “{” (<résultat>) “|” <formule> “}”

<résultat> : := <variable>.<attribut> | <résultat>, <résultat>

<formule> : := <quantification> <formule libre> | <formule libre>

<quantification> : := ∀ <variable> | ∃ <variable>

| <quantification> <quantification>

<formule libre> : := <formule atomique>

| <formule libre> ∧ <formule atomique>

| <formule libre> ∨ <formule atomique>

| <formule libre> ⇒ <formule libre>

| ¬ <formule libre>

| (<formule libre>)

<formule atomique> : := <predicat extensionnel> (<variable>)

| <terme> <comparateur> <terme>

<terme> : := <variable>.<attribut> | <constante>

<comparateur> : := = | < | ≤ | > | ≥ | ≠

Figure V.16 — BNF du calcul relationnel de tuples

5.2 Quelques exemples de calcul de tuple

Nous exprimons maintenant en calcul relationnel de tuples les questions exprimées ci-dessus en calcul de domaines sur la base des produits.

(Q1) Donner la liste des noms et des couleurs de tous les produits :

{(P.NOMP, P.COULEUR) | PRODUIT(P)}

(Q2) Donner les noms et les quantités en stock des produits de couleur rouge :

{(P.NOMP, P.QTES) | PRODUIT(P) ∧ P.COULEUR = "ROUGE"}

(Q3) Donner pour chaque produit en stock, le nom du fournisseur associé :

{(P.NOMP, A.NOMF) | PRODUIT(P)∧ ACHAT(A)
∧ P.NPRO=A.NPRA}

(Q4) Donner, pour chaque produit en stock en quantité supérieure à 100 et de couleur rouge, les couples nom de fournisseurs ayant vendu ce type de produit et nom de client ayant acheté ce type de produit, avec le nom du produit :

{(P.NOMP, A.NOMF, V.NOMC) | PRODUIT(P) ∧ ACHAT(A)
∧ VENTE(V) ∧ P.QTES > 100 ∧ P.COULEUR = "Rouge" ∧
P.NPRO = V.NPRV ∧ P.NPRO = A.NPRA}

(Q5) Donner les noms des clients ayant acheté au moins un produit de couleur verte :

$$\{(V.NOMC) \mid VENTE\ (V) \wedge \exists P\ (PRODUIT\ (P) \wedge\ V.NPRV = P.NPRO \wedge\ P.COULEUR = \text{"Verte"})\}$$

(Q6) Donner les noms des clients ayant acheté tous les produits stockés :

$$\{(V_1.NOMC) \mid VENTE\ (V_1) \wedge \forall P\ (PRODUIT\ (P) \Rightarrow (\exists V_2\ (VENTE(V_2) \wedge V_2.NPRV = P.NPRO \wedge V_2.NOMC = V_1.NOMC\ )))\}$$

(Q7) Donner les noms des produits fournis par tous les fournisseurs et achetés par au moins à un client :

$$\{(P.NOMP) \mid PRODUIT(P) \wedge (\forall A_1\ (ACHAT(A_1) \Rightarrow (\exists A_2\ (ACHAT(A_2) \wedge A_2.NOMF = A_1.NOMF \wedge A_2.NPRA = P.NPRO))) \wedge\ (\exists V\ (VENTE(V) \wedge V.NPRV = P.NPRO))\}$$

Les questions (Q6) et (Q7) démontrent la difficulté d'utilisation des quantificateurs "quel-que-soit" et "il existe". En général, toute variable apparaissant dans la réponse doit être libre dans la condition. Les "il existe" ne sont utiles qu'à l'intérieur des quantifications universelles.

6. LES TECHNIQUES D'INFÉRENCE

La logique permet la déduction. La déduction, principe de l'intelligence, permet de prouver des théorèmes à partir d'axiomes. Les systèmes de bases de données déductifs sont fondés sur l'inférence. Dans cette section, nous rappelons les principes fondamentaux des techniques de déduction. Ces résultats sont supposés connus lors de l'étude des bases de données déductives, dans la quatrième partie de cet ouvrage.

6.1 Principe d'un algorithme de déduction

Un algorithme de déduction est une procédure pour prouver une formule T à partir d'un ensemble de formules {A1, A2,…An} connues comme vraies. T est le théorème à démontrer. A1, A2,…An sont les axiomes. L'existence d'une preuve de T à partir de A1, A2,…An est formellement notée {A1,A2,…An} |== T.

> **Notion V.10 : Déduction (*Deduction*)**
>
> Procédure permettant de prouver un théorème à partir d'un ensemble d'axiomes connus comme vrais sur tout domaine de discours considéré.

Par exemple, à partir des axiomes :

DIRIGE(pierre, marie)

DIRIGE(marie, julie)

$\forall x\ \forall y\ \forall z$ (DIRIGE(x,y) $\wedge$ DIRIGE(y,z) $\Rightarrow$ DIRIGE(x,z))

on aimerait déduire le théorème:

DIRIGE(pierre, julie).

Pour prouver un théorème, un algorithme de déduction dérive à partir des axiomes une séquence de formules dont le théorème est la dernière en utilisant des **règles d'inférence**. Une règle d'inférence est une règle permettant de générer une formule à partir de deux ou plus. Une règle d'inférence correcte permet de générer une formule valide (vraie sur les univers de discours considérés) à partir de formules valides.

Deux règles d'inférences bien connues sont :

- **Le modus ponens.** Il permet de générer P à partir des deux formules F et F $\Rightarrow$ P. Intuitivement, cela signifie que si F et F implique P sont prouvées, alors P est prouvée. On écrit formellement :

$$\frac{F \qquad F \Rightarrow P}{P}$$

- **La spécialisation.** Elle permet de générer F(a) à partir de la formule $\forall x F(x)$. Intuitivement, cela signifie que si F(x) est prouvée pour toute valeur de x, alors F(a) est prouvée. On écrit formellement :

$$\frac{\forall x \ F(x)}{F(a)}$$

Il existe une règle d'inférence générale pour les formules du premier ordre en forme clausale. Cette règle génère par application récursive toutes les formules qui peuvent être déduites à partir de deux axiomes sous forme de clauses. Il s'agit de la **règle d'inférence de Robinson** [Robinson65]. Elle s'appuie sur **l'algorithme d'unification** qui permet de comparer deux formules atomiques.

6.2 Algorithme d'unification

La capacité à décider si deux formules atomiques peuvent être identifiées par substitution de paramètres est centrale à la plupart des méthodes de déduction. Une **unification** de deux formules atomiques consiste à les rendre identiques par remplacement de variables dans une formule par des termes de l'autre.

> **Notion V.11 : Unification (*Unification*)**
>
> Remplacement de variables dans une formule atomique par des termes (constantes, autres variables, fonctions appliquées à des constantes ou des variables) de manière à la rendre identique à une autre formule atomique.

Deux formules atomiques L1(t1,t2,…tn) et L2(s1,s2,…,sn) ne sont pas toujours unifiables. Un algorithme d'unification décide si les deux formules peuvent être identifiées par une substitution de variable valide (renommage ou assignation de valeur). Un tel algorithme (voir figure V.17) retourne :

- SUCCES et une substitution de variable minimale si les deux formules peuvent être identifiées en appliquant la substitution ;

- ECHEC s'il est impossible de les rendre identiques par une substitution de terme unique à chaque variable.

L'algorithme est récursif : il exploite les termes à partir du début en unifiant tête et queue successivement. Il existe beaucoup d'algorithmes d'unification, celui-là n'est qu'indicatif.

Quelques exemples simples permettent d'illustrer l'algorithme. On note {x/t1; y/t2; …} la substitution qui remplace x par t1, y par t2, etc. Les formules DIRIGE(pierre,marie) et DIRIGE(x,y) sont simplement unifiables par la substitution {x/pierre; y/marie}. Les formules DIRIGE(chef(x),pierre) et DIRIGE(chef(chef(pierre)),y) sont unifiables par la substitution {x/chef(pierre); y/pierre}. Les formules DIRIGE(chef(x),x) et DIRIGE(chef(chef(x)),x) ne sont pas unifiables: il faudrait que x devienne à la fois chef(x) et x, ce qui est syntaxiquement impossible.

```
Bool Function Unifier(L1, L2, S) { ; // Unification de L1 et L2
    S := ϕ ; // S est la substitution résultat en cas de succès
    Si (L1 est un atome) alors { // unification d'atome
        Si L1 = L2 alors Retour(Succès)
        Si L1 est une variable alors
            Si L1 ∈ L2 alors Retour(Echec)
            Sinon { S : = S ∪ [L1/L2]
                Retour(Succès) } ;
    Si (L2 est un atome) alors { … idem avec L2 } ;
    M1 = Elément suivant (L1) ; // prendre éléments suivants
    M2 = Elément suivant (L2) ;
    Suite1 = unifier(M1, M2, S1) ; // tenter unification
    Si (Suite1 = Echec) alors Retour(Echec) ;
    N1 = [Reste(L1)/S1)] ; // substituer si succès
    N2 = [Reste(L1)/S1)] ;
    Suite2 = unifier(N1, N2, S2) ; // unifier le reste
    Si (Suite2 = Echec) alors Retour(Echec) ;
    S = S1 ∪ S2 ; // composer les substitutions
    Retour(Succès) ; }
```

Figure V.17 — Un algorithme d'unification

6.3 Méthode de résolution

La méthode de résolution est très simplement basée sur la règle d'inférence de Robinson. Cette règle s'énonce comme suit. Soient C1 et C2 deux clauses de la forme :

$C1 = F1 \vee L1$

$C2 = L2 \Rightarrow F2$.

De plus, moyennant une substitution s, L1 et L2 sont unifiables (on note L1[s] = L2[s]). La règle de Robinson permet d'inférer la clause $F1[s] \vee F2[s]$; cette nouvelle clause obtenue par disjonction de F1 et F2 et application de la substitution s est appelée **résolvant** de C1 et C2. Formellement, la règle de Robinson s'écrit :

$$\frac{F1 \vee L1 \quad L2 \Rightarrow F2 \quad L1[s] = L2[s]}{F1[s] \vee F2[s]}$$

En remarquant que $L2 \Rightarrow F2$ peut aussi s'écrire $\neg L2 \vee F2$, puis en remplaçant le « ou » par + et la négation par -, on obtient :

$$\frac{F1 + L1 \quad F2 - L2 \quad L1[s] = L2[s]}{F1[s] + F2[s]}$$

On voit alors que la règle de Robinson permet finalement de réaliser l'addition de clauses avec suppression des parties s'annulant moyennant une substitution s. Il s'agit de la règle de base du calcul symbolique. Elle a une propriété remarquable de complétude : toute formule pouvant être démontrée à partir d'un ensemble d'axiomes se déduit par applications successives de la règle de Robinson aux axiomes et aux résolvants obtenus.

A partir de cette règle d'inférence est construite la **méthode de résolution**. Cette méthode permet de prouver un théorème à partir d'axiomes non contradictoires. C'est une méthode par l'absurde qui consiste à partir des axiomes et de la négation du théorème et à prouver qu'il y a contradiction. Donc, sous l'hypothèse de non-contradiction des axiomes, c'est que le théorème est valide (car son contraire contredit les axiomes).

En résumé, la méthode procède comme suit :

1. Mettre les axiomes et la négation du théorème ($\neg T$) sous forme de clauses.

2. Ajouter récursivement les résolvants que l'on peut obtenir en appliquant la règle d'inférence de Robinson à l'ensemble des clauses jusqu'à obtention de la clause vide.

La clause vide (tout s'est annulé) ne peut jamais être satisfaite (son modèle est vide) ; par suite, c'est que les axiomes contredisent la négation du théorème. Donc celui-ci est démontré. Il a été démontré que si une preuve du théorème existe, la méthode de résolution se termine et la trouve. Si aucune preuve n'existe, la méthode peut se perdre dans des univers infinis et boucler (de plus en plus de clauses sont générées). La logique du premier ordre est semi-décidable. Pour un approfondissement de cette méthode, consultez [Manna85].

Nous nous contenterons d'illustrer la méthode par un arbre de preuve (figure V.18). Soit à prouver le théorème DIRIGE(pierre, julie) à partir des axiomes non contradictoires :

(A1) DIRIGE(pierre, marie),

(A2) DIRIGE(marie, julie),

(A3) $\forall x\ \forall y\ \forall z$ (DIRIGE(x,y) $\wedge$ DIRIGE(y,z) $\Rightarrow$ DIRIGE(x,z)).

La négation du théorème est ¬DIRIGE(pierre,julie). Les deux premiers sont des clauses. Le troisième se met simplement sous la forme de la clause :

(A'3) DIRIGE(x,y) $\wedge$ DIRIGE(y,z) $\Rightarrow$ DIRIGE(x,z)

qui s'écrit encore :

(A''3) ¬DIRIGE(x,y) $\vee$ ¬DIRIGE(y,z) $\vee$ DIRIGE(x,z).

L'arbre de preuve (encore appelé arbre de réfutation) représenté figure V.18 montre des unifications et additions successives de clauses qui conduisent à la clause vide. Il permet donc par applications successives de la règle d'inférence de Robinson (chaque nœud R non initial dérive de deux nœuds précédents N1 et N2) de tirer le résolvant vide et ainsi de démontrer que Pierre dirige Julie.

<VOIR Maîtriser les BD Fig. VIII.2 p. 295>

Figure V.18 — Exemple d'arbre de preuve

7. CONCLUSION

Nous avons dans ce chapitre rappelé les concepts de base de la logique du premier ordre. Une base de données peut être vue comme l'interprétation d'un langage logique. Cette vision est limitée et nous irons beaucoup plus loin avec les bases de données déductives, traitées dans la 4e partie de cet ouvrage.

Quoi qu'il en soit, la logique constitue une bonne base pour comprendre les bases de données, en particulier les BD relationnelles. Une BD relationnelle est un ensemble de tables donnant les extensions valides des prédicats. Le calcul de tuples et le calcul de domaines sont des formalisations logiques des langages d'interrogation des bases de données relationnelles. Nous allons étudier le modèle relationnel indépendamment de la logique dans la partie qui suit, mais il ne faut pas perdre de vue que la logique est son fondement.

8. BIBLIOGRAPHIE

[Ceri91] Ceri S., Gottlob G., Tanca L., *Logic Programming and Databases*, Surveys in Computer Sciences, Springer Verlag, 1990.

Un livre fondamental sur le modèle logique. Le livre introduit Prolog comme un langage d'interrogation de données, les bases de données relationnelles vues d'un point de vue logique, et enfin les couplages de Prolog et des bases de données. Dans une deuxième partie, Datalog et ses fondements sont présentés. La troisième partie est consacrée aux techniques d'optimisation de Datalog et à un survol des prototypes implémentant ces techniques.

[Clark78] Clark C. « Negation as failure » in *Logic and databases*, Edité par Gallaire et Minker, Plenum Press, New York, 1978.

Un article de base sur la négation en programmation logique. Il est proposé d'affirmer qu'un fait est faux s'il ne peut être démontré vrai (négation par échec). Cela conduit à interpréter les règles comme des équivalences : "si" peut être lu comme "si et seulement si" à condition de collecter toutes les règles menant à un même but en une seule.

[Clocksin81] Clocksin W.F., Mellish C.S., *Programming in Prolog*, Springer Verlag, Berlin-Heidelberg-New York, 1981.

Un livre de base sur le langage Prolog. Prolog utilise des bases de faits en mémoire qui sont similaires aux bases logiques décrites dans ce chapitre. En plus, Prolog gère la déduction.

[Codd72] Codd E.F., « Relational Completness of Data Base Sublanguages », In *Data Base Systems, Courant Computer Science Symposia Series*, N° 6, Prentice-Hall, 1972.

Cet article introduit la notion de complétude pour un langage d'interrogation de bases de données relationnelles. Il donne la définition du calcul relationnel de tuple et de l'algèbre relationnelle. Il démontre que l'algèbre est aussi puissante que le calcul en donnant un algorithme de traduction de calcul en algèbre. Codd argumente aussi sur les avantages du calcul comme base d'un langage utilisateur.

[Gallaire78] Gallaire H., Minker J., *Logic and Databases,* Plenum Press, 1978.

Le premier livre de base sur la logique et les bases de données. Ce livre est une collection d'articles présentés à Toulouse lors d'un "workshop" tenu en 1977 sur le sujet.

[Gallaire81] Gallaire H., Minker J., Nicolas J.M., *Advances in database theory*, Vol. 1, Plenum Press, 1981.

Le second livre de base sur la logique et les bases de données. Ce livre est une collection d'articles présentés à Toulouse lors d'un "workshop" tenu en 1979 sur le sujet.

[Gallaire84] Gallaire H., Minker J., Nicolas J.M., « *Logic and databases: a deductive approach* », ACM Computing Surveys, Vol. 16, N° 2, juin 1984.

Un article de synthèse sur les bases de données et la logique. Différents types de clauses (fait positif ou négatif, contrainte d'intégrité, loi déductive) sont isolés. L'interprétation des bases de données comme un modèle ou comme une théorie de la logique est discutée. Les différentes variantes de l'hypothèse du monde fermé sont résumées.

[Lloyd87] Lloyd J., *Foundations of logic programming*, 2e édition, Springer Verlag Ed., 1987.

Le livre de base sur les fondements de la programmation logique. Les différentes sémantiques d'un programme logique sont introduites. Les techniques de preuve de type résolution avec négation par échec sont développées.

[Manna85] Manna Z., Waldinger R., *The Logical Basis for Computer Programming*, Vol. 1, 618 pages, Addison-Wesley, 1985.

Le livre de base sur la logique. La syntaxe, la sémantique et les méthodes de preuves pour la logique du premier ordre sont développées. La logique du deuxième ordre, où des variables peuvent représenter des prédicats, est aussi étudiée.

[Merrett84] Merrett T.H., *Relational Information Systems*, Prentice Hall, 1984, chapitre 5.

Un bon livre très synthétique sur les bases de données, avec de nombreux exemples pratiques. La relation composant-composé (et plus généralement l'application Facture de Matériel —"Bill of Material") est introduite.

[Nilsson80] Nilsson N., *Principles of Artificial Intelligence*, Tioga Publication, 1980.

Un des livres de base de l'intelligence artificielle. Les systèmes de règles de production pertinents aux bases de données déductives sont particulièrement développés.

[Reiter78] Reiter R., « On closed world data bases », in *Logic and databases*, Edité par Gallaire et Minker, Plenum Press, New York, 1978.

L'article de base sur l'hypothèse du monde fermé.

[Reiter84] Reiter R., « Towards a Logical Reconstruction of Relational Database Theory », in *On Conceptual Modelling*, pp. 191-234, Springer-Verlag Ed., 1984.

Une redéfinition des bases de données relationnelles en logique. Les différents axiomes nécessaires à l'interprétation d'une base de données comme une théorie en logique sont présentés: fermeture des domaines, unicité des noms, axiomes d'égalité, etc. Les calculs relationnels sont unifiés et généralisés.

[Robinson65] Robinson J.A., « A Machine oriented Logic based on the Resolution Principle », *Journal of the ACM*, 12, 1965.

L'article introduisant la méthode de résolution.

[Ullman88] Ullman J.D., *Principles of Database and Knowledge-base Systems*, Vol. I et II, Computer Science Press, 1988.

Deux volumes très complets sur les bases de données, avec une approche plutôt fondamentale. Jeffrey Ullman détaille tous les aspects des bases de données, depuis les méthodes d'accès jusqu'aux modèles objets en passant par le modèle logique. Les livres sont très centrés sur une approche par la logique des bases de données. Les calculs de tuples et domaines, les principaux algorithmes d'accès, d'optimisation de requêtes, de concurrence, de normalisation, etc. sont détaillés.

[Zloof77] Zloof M., « Query-by-Example: A Data Base Language », *IBM Systems Journal*, Vol. 16, N° 4, 1977, pp. 324-343.

Cet article présente QBE, le langage par grille dérivé du calcul de domaines proposé par Zloof, alors chercheur à IBM. Ce langage bi-dimensionnel est aujourd'hui opérationnel en sur-couche de DB2 et aussi comme interface externe du système Paradox de Borland. Zloof discute aussi des extensions bureautiques possibles, par exemple pour gérer le courrier (OBE).