UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# Estudo e Construção de um Sistema Gerenciador de Banco de Dados Dedutivo

por

FABIANE BIZINELLA NARDON

Dissertação submetida à avaliação, como requisito parcial
para a obtenção do grau de
Mestre em Ciência da Computação

Prof. José Mauro Volkmer de Castilho
Orientador

Porto Alegre, julho de 1996

**CIP - CATALOGAÇÃO NA PUBLICAÇÃO**

Nardon, Fabiane Bizinella

Estudo e Construção de um Sistema Gerenciador de Banco de Dados Dedutivo / Fabiane Bizinella Nardon. — Porto Alegre: CPGCC da UFRGS, 1996.

146 p.: il.

Dissertação (mestrado) — Universidade Federal do Rio Grande do Sul. Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, BR-RS, 1996. Orientador: Castilho, José Mauro Volkmer de.

1. Banco de dados. 2. Banco de dados dedutivo. I.Castilho, José Mauro Volkmer de. II. Título.

*Para meu pai e minha mãe*

# Agradecimentos

Ao Prof. Dr. José Mauro Volkmer de Castilho, por ter aceito ser meu orientador, pela amizade, incentivo e excelente orientação, sem as quais este trabalho jamais seria possível.

À Profa. Dra. Beatriz de Faria Leão, cujas excelentes contribuições foram decisivas para a conclusão deste trabalho.

A Eugênio Zimmer Neves, pela solicitude com que forneceu as informações sobre o sistema Aleph.

A Dirceu Sanocki, que auxiliou na implementação do protótipo, e aos demais membros do Grupo de Bancos de Dados Inteligentes, que, além de se tornarem grandes amigos, contribuíram com sugestões que enriqueceram este trabalho.

Aos professores e funcionários do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal do Rio Grande do Sul, pela competência com que realizam suas atividades.

Ao CNPq pelo financiamento deste trabalho.

Ao meu pai Flávio, à minha mãe Maria Helena e às minhas irmãs Fernanda e Flávia, pelo apoio nos momentos decisivos e por, apesar de estarem distantes, estarem também sempre presentes.

Aos meus colegas do CPGCC, que se revelaram amigos maravilhosos nestes dois anos de convivência diária e que são o resultado mais importante deste curso de mestrado.

# Sumário

# Lista de Figuras

# Lista de Abreviaturas

| | |
|---|---|
| BD | Banco de Dados |
| BDD | Banco de Dados Dedutivo |
| BDE | Banco de Dados Extensional |
| BDI | Banco de Dados Intensional |
| HMF | Hipótese do Mundo Fechado |
| MLM | Medical Logic Modules |
| SBD | Sistema de Banco de Dados |
| SGBD | Sistema Gerenciador de Banco de Dados |
| SGBDD | Sistema Gerenciador de Banco de Dados Dedutivo |

# Resumo

Este trabalho apresenta o estudo e a construção de um Sistema Gerenciador de Bancos de Dados Dedutivos. Um Banco de Dados Dedutivo (BDD) é um Banco de Dados que, além de sua parte tradicional, ou seja, as informações contidas nas relações básicas, que são explicitamente inseridas, possui um conjunto de regras dedutivas que permite derivar novas informações a partir das relações básicas.

Neste trabalho, as deficiências da linguagem de consulta Datalog foram identificadas e, com o objetivo de obter uma linguagem que atenda melhor algumas das necessidades de aplicações do mundo real, foram propostas extensões ao Datalog, que deram origem à linguagem DEDALO.

As atualizações sobre Bancos de Dados Dedutivos também foram estudadas, sendo identificados dois problemas: o primeiro se refere à necessidade de propagar modificações sobre as relações básicas para as relações derivadas materializadas; o segundo problema diz respeito às atualizações sobre as relações derivadas, que devem ser traduzidas em atualizações sobre as relações básicas, para que a atualização pretendida se torne visível na relação derivada. Para o primeiro problema, métodos de propagação foram estudados, analisados e implementados. Para o segundo, foram estudados, analisados, propostos e implementados métodos que realizam a tradução das atualizações.

Restrições de integridade em Bancos de Dados Dedutivos também foram estudadas, sendo propostos métodos eficientes de detecção de violações de integridade e de reparos de transações que violam as restrições definidas no sistema.

Os estudos realizados deram origem ao Sistema Gerenciador de Banco de Dados Dedutivo DEDALO, um protótipo que implementa a nova linguagem proposta como uma extensão do Datalog, os métodos de propagação de atualizações para relações derivadas materializadas, as técnicas de tradução de atualizações sobre relações derivadas, os métodos de detecção de violação de restrições de integridade e as técnicas de reparo das transações que as violam.

O Sistema DEDALO é composto de quatro ferramentas: o *Gerenciador de Regras*, onde as regras de derivação e as restrições de integridade são definidas; a *Interface Interativa*, utilizada para submeter consultas ad hoc e solicitações de atualização sobre o sistema; dois novos *Componentes Delphi*, que são duas novas classes criadas para o ambiente de desenvolvimento de aplicações Delphi, que foi utilizado na implementação do protótipo, e são utilizadas para a criação das aplicações sobre o Sistema DEDALO; e o *Tradutor de Sentenças DEDALO/SQL-ANSI*, que traduz as sentenças da linguagem proposta para sentenças SQL-ANSI que serão submetidas ao Sistema Gerenciador do Banco de Dados.

A adequabilidade das soluções estudadas e implementadas no sistema desenvolvido pôde ser comprovada através da implementação de uma aplicação real utilizando o Sistema DEDALO.

**Palavras-chave:** Bancos de Dados, Bancos de Dados Dedutivos

**Title**: "Study and Construction of a Deductive Database Management System"

# Abstract

This work presents the study and construction of a Deductive Database Management System. A Deductive Database (BDD) is a Database that, beyond its traditional part, i.e., the informations contained in basic relations, that are explicitly introduced, has a deductive rules set, which permit to derive new informations from the basic relations.

In this work, the deficiencies of Datalog language were identified and, with the goal of obtaining a language which could better support some real world applications requirements, extensions to Datalog are proposed, which originate the DEDALO language.

Deductive Databases updates were also studied and two problems were identified: the first one refers to the need of propagating updates over basic relations to the materialized derived relations; the second problem refers to the updates over derived relations, which must be translated in updates over basic relations, in order that the requested update become visible in the derived relation. For the first problem, propagation methods were studied, analyzed and implemented. For the second, methods that do the updates translation were studied, analyzed, proposed and implemented.

Integrity Constraints in Deductive Database Systems were also studied and methods were proposed to the efficient integrity constraints violation detection and repairing, for transactions that violate system defined constraints.

The studies originate the DEDALO Deductive Database Management System, a prototype that implements the new language, proposed as a Datalog extension. The prototype also contains the implementation of updates propagation methods for materialized derived relations, the translation techniques of updates over derived relations, the integrity constraints violation detection methods and the repairing techniques for the transactions that violate the constraints.

The DEDALO System is composed of four tools: the *Rules Manager*, where derivation rules and integrity constraints are defined; the *Interactive Interface*, used for submit ad hoc queries and updates request to the system; two new *Delphi Components*, which are two new classes created for the application development environment Delphi, which was used in the prototype implementation, and are used in the applications creation over the DEDALO System; and the *DEDALO/SQL-ANSI*

*Statements Translator*, which translate the statements of the proposed language to SQL-ANSI statements, which will be submitted to the Database Management System.

The adequacy of the solutions studied and implemented in the prototype system was verified by the implementation of an application using the DEDALO System.

**Keywords**: Database, Deductive Database

# 1 Introdução

Sistemas de Bancos de Dados Dedutivos [CAS 86, CER 90, GAL 84, KOR 94, LOB 93, NAV 89, RAM 95] são sistemas capazes de fornecer todos os serviços de um Sistema Gerenciador de Banco de Dados tradicional, como controle de concorrência, recuperação em caso de falha, independência de dados, e que permitem, adicionalmente, a dedução de novas informações a partir daquelas explicitamente inseridas no Banco de Dados. A dedução de novas informações é realizada por um conjunto de regras dedutivas que fazem parte do esquema do Banco de Dados Dedutivo. As relações que contêm as informações explicitamente inseridas, são chamadas de *relações básicas* e as relações que são deduzidas pelas regras dedutivas são chamadas de *relações derivadas*. Os Sistemas de Bancos de Dados Dedutivos utilizam uma linguagem de consulta baseada em lógica, que, além de ser uma linguagem declarativa, permite que se expressem consultas recursivas.

O interesse pelo tema Bancos de Dados Dedutivos pode ser medido pelo grande número de trabalhos e sistemas deste tipo desenvolvidos nos últimos anos. Entre os Sistemas de Bancos de Dados Dedutivos existentes, pode-se citar CORAL [RAM 92], LDL [CHI 90], LogiQuel [MAR 91], Aditi [VAG 91], LQL [SHA 88], RDL1 [KIE 90], Glue-Nail [PHI 91], Rock & Roll [BAR 94], ConceptBase [JAR 94, JEU 93] e Chronolog [BÖH 94], desenvolvidos em ambientes universitários, e o Sistema DECLARE [KIE 93], desenvolvido como um Sistema de Banco de Dados Dedutivo comercial.

Apesar do grande número de pesquisas na área e de freqüentemente serem citados como uma tecnologia promissora, os Sistemas de Bancos de Dados Dedutivos não são muito utilizados no desenvolvimento de aplicações do mundo real [KIE 90]. A área é freqüentemente considerada mais teórica do que prática. Existem duas razões que explicam isto:

1. A linguagem de consulta utilizada, geralmente baseada no Datalog, que é uma evolução do Prolog, criada especificamente para uso com Bancos de Dados, não possui características que a tornem uma linguagem adequada para utilização em aplicações práticas. Particularmente, presença de negação, operações aritméticas, operações de comparação e funções agregadas, são características desejáveis em uma linguagem de consulta e que estão ausentes no Datalog puro e em muitas das linguagens de consulta utilizadas em implementações de Bancos de Dados Dedutivos. A forma com que os atributos de uma relação são referenciados, pela sua posição e não pelo seu nome, também é uma característica incômoda destas linguagens. Além disso, elas geralmente só atendem

a consultas e não são capazes de expressar atualizações, característica essencial em aplicações reais;

2. A implementação de muitos dos Sistemas de Bancos de Dados Dedutivos, utiliza um Sistema Gerenciador de Banco de Dados específico, ao qual é adicionada capacidade de tratar regras de dedução. Este tipo de implementação não permite que se mantenha a base de dados já instalada, tendo-se que convertê-la para o SGBD sobre o qual o Sistema de Banco de Dados Dedutivo foi criado.

As atualizações em um Sistema de Banco de Dados Dedutivo necessitam de tratamento adicional àquele provido por um Sistema de Banco de Dados tradicional. Em particular, o problema das atualizações sobre relações derivadas, que está fortemente relacionado ao problema das atualizações sobre visões em Sistemas de Bancos de Dados relacionais, e o problema da propagação de atualizações para relações derivadas materializadas, merecem atenção especial.

## 1.1 Objetivos do trabalho

O objetivo principal deste trabalho é estudar a aplicabilidade dos Sistemas de Bancos de Dados Dedutivos em situações práticas, propondo uma linguagem de consulta que atenda as necessidades destas situações e soluções para os problemas que ainda existem na área. Os resultados deste estudo serão utilizados na construção de um protótipo que deverá possuir as funcionalidades necessárias para atender aplicações do mundo real. Para atingir este objetivo maior, com este trabalho pretende-se também:

1. Definir uma nova linguagem que estenda o poder expressivo da linguagem Datalog e seja adequada para uso em situações práticas;

2. Estudar o problema das atualizações em Bancos de Dados Dedutivos, propondo e avaliando soluções que o resolvam com confiabilidade e eficiência;

3. Prover mecanismos para especificação de restrições de integridade bem como métodos de verificação de integridade eficientes;

4. Prover mecanismos de reparo para transações que violam as restrições de integridade definidas no sistema;

5. Construir uma ferramenta que auxilie na criação e manutenção da base de regras e das restrições de integridade;

6. Implementação de um protótipo de um Sistema Gerenciador de Banco de Dados Dedutivo em que serão postos em prática os estudos realizados. O sistema implementado deverá poder ser utilizado em conjunto com um grande número de SGBDs comerciais, para que as bases de dados já instaladas possam ser aproveitadas.

Para que a adequabilidade das soluções criadas sejam testadas, uma aplicação real será implementada utilizando o protótipo desenvolvido.

## 1.2 Organização do texto

Este trabalho está dividido em dez capítulos. Este primeiro capítulo introduz o trabalho e seus objetivos. O capítulo 2 apresenta uma breve introdução aos Sistemas de Bancos de Dados Dedutivos, algumas definições e conceitos básicos e uma discussão sobre o Datalog e suas deficiências. O capítulo 3 apresenta a linguagem DEDALO, uma extensão do Datalog criada para suprir as deficiências citadas no capítulo 2. Os capítulos 4 e 5 discutem o problema das atualizações sobre Bancos de Dados Dedutivos, sendo que o capítulo 4 aborda o problema da necessidade de propagação das atualizações sobre relações básicas para as relações derivadas materializadas, e o capítulo 5 discute o problema das atualizações sobre relações derivadas, analisando e propondo soluções. O capítulo 6 discute a introdução de restrições de integridade em um Sistema de Banco de Dados Dedutivo, propondo métodos para sua verificação e para o reparo das transações que violam estas restrições. O capítulo 7 apresenta uma forma de gerenciamento das transações, onde as atualizações sobre o Banco de Dados Dedutivo são tratadas, as restrições de integridade são verificadas e, se necessário, reparadas. O capítulo 8 apresenta o Sistema DEDALO, que é o protótipo que implementa as idéias apresentadas neste trabalho. O capítulo 9 apresenta uma aplicação real e sua implementação utilizando o protótipo desenvolvido. Por fim, o capítulo 10 conclui este trabalho.

# 2 Introdução aos Sistemas de Bancos de Dados Dedutivos

Este capítulo apresenta uma breve introdução aos Sistemas de Bancos de Dados Dedutivos. Na seção 2.1 é feito um pequeno histórico dos BDD e apresentado seu conceito. Na seção 2.2, os conceitos de termo, átomo, literal, fato, regra e regra recursiva são introduzidos. Por fim, a seção 2.3 apresenta a linguagem Datalog e uma breve discussão sobre suas principais deficiências.

## 2.1 Sistemas de Bancos de Dados Dedutivos

Sistemas de Bancos de Dados Dedutivos têm sua origem ligada aos estudos sobre a integração de lógica com Bancos de Dados [GAL 78, GAL 84, ULL 85]. Estes estudos procuravam utilizar linguagens de programação em lógica como linguagens de consulta em Sistemas de Banco de Dados.

Nas primeiras tentativas de implementação, a linguagem baseada em lógica escolhida foi o Prolog, por ser a linguagem de programação em lógica mais utilizada e conhecida. O objetivo era adaptar a visão de pequeno volume de dados do Prolog à visão de grande volume de dados dos Sistemas de Bancos de Dados. Estas primeiras tentativas não obtiveram sucesso em razão de algumas características indesejáveis do Prolog, como a ineficiência em tratar grandes volumes de dados e a importância na ordem das regras, o que não fazia do Prolog uma linguagem totalmente declarativa como se desejava. Isto levou à criação da linguagem de consulta Datalog, descrita na seção 2.3, que surgiu como uma alternativa ao uso do Prolog como linguagem de consulta para Bancos de Dados.

Na década de 80, os Sistemas de Bancos de Dados Dedutivos, e em particular o processamento de consultas recursivas, tornou-se uma área muito ativa, com o início de três projetos principais: o projeto Nail! [MOR 86] em Stanford, o projeto LDL [CHI 90] no Microeletronics and Computer Technology Corporation (MCC) em Austin, e o projeto de Banco de Dados Dedutivo no ECRC [VIE 86, VIE 87]. Estes projetos trouxeram significantes contribuições para a pesquisa na área e levaram à construção de protótipos.

Um Banco de Dados Dedutivo (BDD) é um Banco de Dados que, além de sua parte tradicional, ou seja, as informações contidas nas relações básicas que são

explicitamente inseridas, possui um conjunto de regras dedutivas que permite derivar novas informações a partir das relações básicas. Assim, um Banco de Dados Dedutivo pode ser dividido em duas partes:

i. O Banco de Dados *Extensional* (BDE): que é formado pelo conjunto de fatos básicos, contidos nas relações base, que foram explicitamente inseridos.

ii. O Banco de Dados *Intensional* (BDI): que é formado pelas informações contidas nas relações derivadas, deduzidas pela aplicação das regras dedutivas sobre o BDE.

O estado do Banco de Dados em um BDD não é formado apenas pelo conteúdo das relações básicas, mas por este e por todos os dados implícitos que podem ser derivados do BDE através das regras dedutivas.

## 2.2 Definições e alguns conceitos básicos

O objetivo desta seção é introduzir algumas definições e conceitos básicos necessários para a compreensão do restante do texto.

Um *termo* é uma variável ou uma constante. Se *f* é um símbolo funcional n-ário, e *t1,...,tn* são termos, então *f(t1,...,tn)* também é um termo.

Se *p* é um símbolo predicativo n-ário e *t1,...,tn* são termos, então *p(t1,...,tn)* é um *átomo*. Um *literal* é um átomo positivo *p(t1,...,tn)* ou um átomo negado *not(p(t1,...,tn))*.

Um *fato* é normalmente representado por um predicado com termos constantes. Um fato equivale a uma tupla do modelo relacional.

Uma *regra* em Bancos de Dados Dedutivos é normalmente escrita com a seguinte notação:

$$p :- q_1,...,q_n.$$

Esta regra pode ser lida como "$q_1$ e $q_2$ e ... e $q_n$ implicam p". *p* é a *cabeça* da regra e $q_1,...,q_n$ é o *corpo* da regra. Tanto *p* como cada um dos $q_i$'s são literais. O predicado da cabeça da regra representa a relação derivada e os predicados do corpo da regra representam relações básicas ou mesmo outras relações derivadas.

Se a relação derivada representada na cabeça da regra aparece como um dos predicados do corpo ou pelo menos um dos predicados do corpo representa uma relação derivada que depende da relação da cabeça para ser computada, diz-se que a regra é *recursiva*. Uma regra recursiva pode ser facilmente identificada observando-se o *grafo de dependências* entre as regras. Um grafo de dependências é um grafo dirigido cujos nodos são as relações que ocorrem nas regras. Se um nodo corresponde à cabeça de uma regra, deve haver um arco que sai de cada nodo que representa uma relação do corpo da regra apontando para o nodo que corresponde à cabeça. Um ciclo

neste grafo identifica uma recursão. Por exemplo, considerando o seguinte conjunto de regras:

*p(X,Y) :- q(X,Y).*

*p(X,Y) :- q(X,Z) , p(Z,Y).*

cujo grafo de dependências é apresentado na figura 2.1. Pode-se notar que existe um ciclo no grafo, pois a relação *p* depende dela mesma. Assim, *p* é uma relação derivada definida por uma regra recursiva.

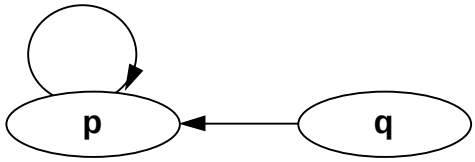


FIGURA 2.1 - Grafo de dependências de um conjunto de regras recursivas

## 2.3 Datalog

As tentativas iniciais de utilização de linguagens de programação baseadas em lógica como linguagem de consulta para Bancos de Dados buscaram adaptar a linguagem Prolog para uso com Sistemas de BD. O Prolog, porém, apresentou uma série de deficiências, entre elas:

? Em Prolog, a ordem das regras tem influência no resultado final, enquanto se desejava uma linguagem de consulta totalmente declarativa, onde a ordem das regras não fosse importante.

? A estratégia de avaliação do Prolog recupera uma tupla por vez, enquanto em Sistemas de Bancos de Dados é mais adequado a recuperação de conjuntos de tuplas por vez.

Em vista disso, foi definida uma nova linguagem baseada em lógica para uso específico com Bancos de Dados, o Datalog [CER 90].

O Datalog é uma linguagem de consulta declarativa baseada em lógica que não possui predicados pré-definidos, negação, disjunção e símbolos funcionais. Em Datalog, a ordem das regras não tem importância.

Uma regra é avaliada derivando o conjunto de todas as constantes possíveis que fazem a cabeça da regra verdadeira, e designando estes valores para a nova relação que é a cabeça da regra. A interpretação de um conjunto de regras Datalog é baseada na *semântica de ponto fixo* [ABI 90, BAN 86]. Nesta semântica, as regras são

aplicadas repetidamente até um ponto fixo ser atingido, isto é, até que o estado do Banco de Dados não possa ser modificado pela aplicação de uma regra

As diversas restrições do Datalog, como a ausência de funções, negação, disjunção, operações aritméticas e operações de comparação, simplificam as formulações teóricas e garantem que toda derivação será sempre finita. No entanto, um Sistema de Banco de Dados construído nas bases desta teoria não atende os requisitos impostos por aplicações do mundo real. Assim, os Sistemas de Bancos de Dados Dedutivos [BAR 94, BÖH 94, CHI 90, JAR 94, KIE 90, MAR 91, PHI 91, RAM 92, SHA 88, VAG 91], em geral, utilizam extensões do Datalog puro [ABI 90,ABI 91] como linguagem de consulta. Uma breve discussão sobre as deficiências do Datalog é apresentada a seguir.

### 2.3.1 Negação

O poder expressivo de regras com apenas literais positivos é excessivamente restrito para as necessidades das aplicações reais. A introdução da negação em regras Datalog permite que se expressem as operações de diferença e divisão de conjuntos da álgebra relacional.

A abordagem geralmente utilizada para computação da negação em Sistemas de Bancos de Dados Dedutivos é a Hipótese do Mundo Fechado (HMF). A HMF [APT 94, GAL 84] diz que fatos não conhecidos verdadeiros são assumidos falsos. Isto é, $not(R(at_1,...,at_n))$ é assumido verdadeiro se e somente se a tupla $<at_1,...,at_n>$ não é encontrada na relação *R*.

A adoção da HMF implica que no momento da avaliação da negação, a relação que corresponde ao literal negado deve conter todas as tuplas válidas, pois a afirmação de que um fato é falso depende de se conhecer todos os fatos verdadeiros. Se o literal negado corresponde a uma relação básica, esta condição é trivialmente satisfeita. Se a relação é derivada, deve-se garantir que esta esteja totalmente computada antes da avaliação da regra. Esta condição pode ser garantida exigindo-se que a negação seja *estratificada* [ABI 88, BÖH 94, KOR 94, RAM 95]. Informalmente, um conjunto de regras é estratificado se pode ser dividido em níveis tais que um predicado somente é usado dentro de uma negação se ele foi definido em um nível mais baixo (chamado *strata*). Na prática, isto evita que a negação esteja envolvida em um ciclo de recursão. O exemplo 2.1 apresenta um conjunto de regras com negação não estatificada.

**Exemplo 2.1:** O seguinte conjunto de regras:

*sadia(X) :- pessoa(X) , not(doentes(X)).*
*doentes(X) :- pessoa(X), not(sadia(X)).*

não é estratificado, pois a negação está envolvida em um ciclo.

Alternativamente à utilização da estratificação na computação da negação, existem abordagens que adotam como válidas as regras *localmente estratificadas*

[CER 90, PRZ 88, RAM 95]. Um conjunto de regras é localmente estratificado para um dado Banco de Dados se, substituindo as variáveis por constantes, em todas as formas possíveis, as regras instanciadas resultantes não possuem negação envolvida em recursão. Esta abordagem é utilizada nos Sistemas de Bancos de Dados Dedutivos Concept-Base [JEU 93] e DECLARE [KIE 93]. O exemplo 2.2 apresenta um conjunto de regras localmente estratificado.

**Exemplo 2.2:** O seguinte conjunto de regras [CER 90]:

*p(b) :- not(p(a)).*
*p(c) :- not(p(b)).*

não é estratificado, mas é localmente estratificado, pois nenhum fato depende dele mesmo.

A estratificação pura, no entanto, é a abordagem mais utilizada por Sistemas de Bancos de Dados Dedutivos, pois as regras estratificadas podem ser avaliadas eficientemente e possuem uma semântica muito intuitiva. Entre os sistemas que utilizam a estratificação estão ADITI [VAG 91], COL [ABI 91], EKS-VI [BAY 93], Hy+ [CON 93] e LOLA [FRE 91].

### 2.3.2 Funções

O valor retornado por uma função arbitrária qualquer é imprevisível. Claramente, a utilização destas funções pode levar a derivações infinitas. O Datalog puro, e diversos outros Sistemas de Bancos de Dados Dedutivos (DBProlog [MAT 89], ConceptBase [JEU 93, JAR 94]), eliminam este problema simplesmente proibindo a utilização de funções.

A utilização de funções agregadas (*count, sum, min, max, avg*), no entanto, é de grande utilidade para um vasto número de aplicações do mundo real. Estas funções podem ser introduzidas no Datalog com sucesso, desde que exista algum mecanismo que garanta sua correta computação.

As funções agregadas são aplicadas sobre um conjunto de tuplas. Para que possam ser corretamente computadas, todo o conjunto deve estar disponível no momento da aplicação do operador agregado. Isto pode ser garantido observando-se a condição de estratificação também para a agregação [RAM 95].

### 2.3.3 Operações aritméticas

Operações aritméticas podem levar a derivações infinitas e não são permitidas em Datalog puro. O exemplo 2.3 demonstra uma situação em que a presença de uma operação aritmética leva a uma derivação infinita:

**Exemplo 2.3:**

*números_naturais(N) :- zero(N).*

*números_naturais(N+1) :- números_naturais(N). ,*

onde *zero* é uma relação básica e *números_naturais* é uma relação derivada. Supondo que a relação básica possua a única tupla:

*zero(0).*

a relação *números_naturais* seria infinita.

Operações aritméticas em ciclos de recursão sempre são uma origem potencial de derivações infinitas. No entanto, existem inúmeras situações em que se pode utilizar operações aritméticas com recursão e obter uma derivação finita. Esta situação é demonstrada pelo exemplo 2.4.

**Exemplo 2.4:**

*números naturais(N) :- zero(N).*

*números_naturais(N+1) :- números_naturais(N) , N < 100.*

onde *zero* é uma relação básica, cujo conteúdo é a tupla:

*zero(0).*

e *números_naturais* é uma relação derivada.

Neste caso, apesar da operação aritmética estar envolvida em um ciclo de recursão, a derivação seria finita. Em vista disso, não é aconselhável proibir operações aritméticas dentro de recursão como forma de assegurar derivações finitas. Isto seria excessivamente restritivo e excluiria da linguagem a capacidade de expressar um grande número de regras que não levariam a computações sem fim.

### 2.3.4 Operações de Comparação

Em Datalog puro, operadores de comparação (igual, diferente, menor, maior, menor ou igual e maior ou igual) não são permitidos. No entanto, estas operações podem ser introduzidas no Datalog sem problemas. Uma única condição deve ser observada: as variáveis envolvidas nas operações de comparação devem estar instanciadas no momento da avaliação da operação.

### 2.3.5 Atributos referenciados por sua posição dentro da relação

Em Datalog, como em Prolog, os atributos de uma relação, que correspondem aos argumentos de um predicado, são referenciados por sua posição dentro do predicado e não pelo seu nome.

Considerando o Banco de Dados do exemplo 2.5:

**Exemplo 2.5:**

Relações Básicas: *pacientes(Código,Nome,Idade,Médico_Responsavel).*
*médicos(Código,Nome,Endereço,Telefone).*
Relação Derivada: *pac_resp(Nome,Médico) :- pacientes(C,Nome,I,M) ,*
*médicos(M,Médico,E,T).*

Para referir-se ao atributo *Nome* da relação *pacientes*, é necessário que se conheça sua posição dentro da relação. Assim, como o nome do paciente é o segundo atributo dentro do predicado, a variável *Nome*, colocada na segunda posição, será instanciada com os valores do atributo *Nome* da relação *pacientes*. Adicionalmente, para cada relação é necessário que todos os atributos sejam citados, ou seja, se uma relação possui *n* atributos, é necessário que toda referência a ela possua *n* variáveis ou constantes, cada uma representando a posição de um atributo, mesmo que se deseje obter o valor de apenas alguns dos atributos[1], como acontece na regra que define a relação derivada *pac_resp*, onde *Código* e *Idade* do paciente e *Endereço* e *Telefone* do médico não seriam necessários.

Em aplicações reais, o Banco de Dados geralmente é formado por muitas relações e cada relação possui muitos atributos. Nestas situações, referir-se aos atributos por sua posição e não pelo seu nome é extremamente desagradável e confuso [ULL 91]. É difícil lembrar-se para cada relação, em que ordem foram inseridos os atributos para que se possa referenciá-los corretamente.

### 2.3.6 Operações de Atualização

O Datalog foi idealizado como uma linguagem de consulta, por isso, não possui sentenças que possam expressar solicitações de atualização no Banco de Dados.

Seria interessante que as operações de atualização pudessem ser expressas também em Datalog para que se pudesse utilizar uma linguagem unificada para todas as operações sobre o Banco de Dados.

---

[1] Alternativamente, como permite o Prolog, nas posições que representam atributos cujo valor não importa, pode-se utilizar uma variável nula (*underline* no Prolog).

## 2.4 Comentários finais

As características dos Sistemas de Bancos de Dados Dedutivos como a capacidade de responder a consultas recursivas, a linguagem de consulta declarativa, a dedução de novas informações, permitem que estes sistemas realizem tarefas não suportadas por Sistemas de Bancos de Dados convencionais. Aplicações como bases de dados científicas, controle de tráfego aéreo, análise exploratória de dados, além das aplicações tradicionais de SBD, são freqüentemente citadas [TSU 91] como aplicações que encontrariam grandes vantagens se resolvidas com um Sistema de Banco de Dados Dedutivo.

O Datalog possui a capacidade de expressar consultas recursivas e é uma linguagem baseada em lógica declarativa. Estas características, embora desejáveis em uma linguagem de consulta, não são suficientes para fazer do Datalog uma linguagem adequada para aplicações do mundo real. A necessidade de suprir as deficiências citadas neste capítulo exige que se proponha uma série de extensões ao Datalog puro. Estas extensões, que fazem parte da linguagem DEDALO[2], utilizada pelo Sistema de Banco de Dados Dedutivo descrito no capítulo 8, são apresentadas no próximo capítulo.

[2]Dedução, Dados e Lógica

# 3 DEDALO : Uma nova linguagem baseada em lógica para BDD

Este capítulo apresenta a linguagem DEDALO (Dedução, Dados e Lógica), utilizada pelo Sistema de Banco de Dados Dedutivo de mesmo nome descrito no capítulo 8. A linguagem DEDALO é uma extensão do Datalog que busca suprir as deficiências do Datalog puro, apresentadas no capítulo 2, com a introdução de negação, funções agregadas, operações aritméticas, disjunção, operações de comparação e operações de atualização. Adicionalmente, foram criadas notações que permitem referenciar os atributos pelo seu nome e não por sua posição e especificar a ordem desejada das tuplas na relação derivada resultante. A linguagem DEDALO permite também raciocínio aproximado, através de adaptações da lógica fuzzy para Bancos de Dados Dedutivos. A BNF da linguagem é apresentada no Anexo 1.

As extensões introduzidas e os mecanismos que garantem a correta computação das regras nesta linguagem serão descritos a seguir.

## 3.1 Referenciando os atributos pelo nome

Na linguagem DEDALO, os atributos de uma relação são referenciados pelo seu nome e não por sua posição dentro da relação. Uma regra de derivação em DEDALO tem a seguinte forma:

$$p(atr_1{:}argc_1,atr_2{:}argc_2,...,atr_n{:}argc_n) \text{ :- } q_1(atr_1{:}arg_1,...,atr_n{:}arg_n) ,$$
$$q_2(atr_1{:}arg_1,...,atr_n{:}arg_n),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n).$$

onde $p(atr_1{:}argc_1,atr_2{:}argc_2,...,atr_n{:}argc_n)$ é a cabeça da regra, $q_1(atr_1{:}arg_1,...,atr_n{:}arg_n),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n)$ é o corpo da regra, sendo que cada um dos $q_i$s representa uma relação básica ou derivada, $atr_i$ é o nome de um atributo da relação, $argc_i$ é uma variável ligada a um atributo do corpo, ou uma constante, ou uma operação aritmética ou uma função agregada e $arg_i$ é uma variável ligada a um atributo, ou uma constante ou uma operação aritmética. Seguindo a notação utilizada pelo Prolog, a vírgula (,) representa uma conjunção. Cada uma das variáveis que aparecem na cabeça da regra devem estar ligadas a um atributo no corpo da regra.

O exemplo 3.1 apresenta a regra de derivação do exemplo 2.5, utilizando a sintaxe da linguagem DEDALO.

**Exemplo 3.1:**

*pac_resp(Nome:N,Médico:M):-pacientes(Nome:N,Médico_responsável:C) ,*
*médicos(Código:C,Nome:M).*

A ordem dos atributos dentro do predicado não é importante e, nas relações do corpo, os atributos que não são necessários não precisam ser citados. Na cabeça da regra, todos os atributos devem ser citados, pois a cabeça representa a relação derivada que será criada. Adicionalmente, se duas ou mais regras possuem a mesma cabeça, o que é interpretado como a união dos conjuntos de tuplas derivadas de cada uma das regras, as cabeças devem ser compatíveis, ou seja, devem possuir o mesmo número de atributos e os atributos devem ser de tipos compatíveis.

A presença de uma constante ligada a um dos atributos de um predicado do corpo da regra representa uma condição que deve ser satisfeita pelas tuplas da relação representada pelo predicado, ou seja, serão selecionadas as tuplas da relação cujo atributo possua valor igual à constante informada.

## 3.2 Introduzindo negação

A negação de um predicado pode ser expressa na linguagem DEDALO através do operador *not*. Uma regra com literais negativos teria a forma:

$$p(atr_1{:}argc_1,atr_2{:}argc_2,...,atr_n{:}argc_n) :- q_1(atr_1{:}arg_1,...,atr_n{:}arg_n) ,...,$$
$$not(q_i(atr_1{:}arg_1,...,atr_n{:}arg_n)),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n).$$

A negação é computada pela Hipótese do Mundo Fechado e as regras devem ser estratificadas.

## 3.3 Introduzindo operações aritméticas

Operações aritméticas são permitidas na linguagem DEDALO tanto na cabeça como no corpo das regras. Uma regra com operações aritméticas teria a forma:

$$p(atr_1{:}argc_1,atr_2{:}op_1?op_2,...,atr_n{:}argc_n) :- q_1(atr_1{:}arg_1,...,atr_n{:}arg_n) ,...,$$
$$q_i(atr_1{:}op_3?op_4,...,atr_n{:}arg_n),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n).$$

onde $op_i$ é uma variável ou uma constante ou uma expressão aritmética e ? é um dos quatro operadores aritméticos permitidos (+,-,* e /).

Operações aritméticas na cabeça e no corpo das regras têm significados diferentes. Na cabeça da regra, o resultado da operação representa o valor que o atributo associado à operação aritmética terá na relação derivada. No corpo da regra,

o resultado da operação representa o valor que o atributo ao qual a operação aritmética está associada deve possuir para que a tupla correspondente seja selecionada.

Para garantir a correta computação da operação aritmética, cada uma das variáveis envolvidas devem estar instanciadas no momento da avaliação da operação.

O exemplo 3.2 demonstra a utilização de operações aritméticas na cabeça de uma regra:

**Exemplo 3.2:**

*PAMs(Nome:X,PAM:(Y+2*Z)/3) :- pacientes(Nome:X,PAS:Y,PAD:Z).*

Nesta regra, a operação aritmética *(Y+2*Z)/3* calculará a pressão arterial média (PAM) dos pacientes, em função da pressão arterial sistólica (PAS) e da pressão arterial diastólica (PAD).

O exemplo 3.3 demonstra a utilização de operações aritméticas no corpo de uma regra:

**Exemplo 3.3:**

*PAM_Corretas(Nome:X):-*
*pacientes(Nome:X,PAS:Y,PAD:Z,PAM:(Y+2*Z)/3).*

Neste exemplo, a relação derivada *PAM_Corretas* conterá todos os nomes de pacientes cuja pressão arterial média foi calculada corretamente em função da pressão arterial sistólica e diastólica.

Como foi discutido no capítulo 2, a introdução de operações aritméticas pode levar a derivações infinitas em alguns casos. Para não restringir o poder expressivo das regras, na linguagem DEDALO, especificar regras que gerem derivações finitas é responsabilidade do usuário.

## 3.4 Introduzindo operações de comparação

Operações de comparação são utilizadas para representar condições que devem ser satisfeitas no corpo das regras. Uma regra com operações de comparação teria a forma:

$p(atr_1{:}argc_1,atr_2{:}argc_2,...,atr_n{:}argc_n)$ :- $q_1(atr_1{:}arg_1,...,atr_n{:}arg_n)$, ..., $q_i(atr_1{:}arg_1,...,atr_n{:}arg_n)$,...,$q_n(atr_1{:}arg_1,...,atr_n{:}arg_n)$, $arg'_1$ ? $arg'_2$.

onde $arg'_i$ é uma variável ligada a um atributo, ou uma constante ou uma expressão aritmética e ? é um dos operadores de comparação permitidos (>, < , =, <>, <=, >=).

O exemplo 3.4 apresenta um regra com uma operação de comparação.

**Exemplo 3.4:**

*PacientesComFebre(Nome:X) :- Pacientes(Nome:X,Temp:Y) , Y > 37.*

Neste exemplo, a relação *PacientesComFebre* receberia os nomes de todos os pacientes cuja temperatura é maior que 37?.

Como nas operação aritméticas, para que as operações de comparação possam ser corretamente avaliadas, é necessário que, no momento da avaliação, as variáveis envolvidas na comparação estejam instanciadas.

## 3.5 Introduzindo funções agregadas

As funções agregadas *sum, count, min, max* e *avg*, respectivamente, soma, contagem, mínimo, máximo e média de um conjunto de tuplas podem ser utilizadas na cabeça das regras. Uma regra contendo funções agregadas teria a forma:

$p(atr_1{:}argc_1,...,atr_i{:}função(var_i),...,atr_n{:}argc_n)$ :- $q_1(atr_1{:}arg_1,...,atr_n{:}arg_n)$ ,
$q_2(atr_1{:}arg_1,...,atr_n{:}arg_n),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n)$.

onde *função* é uma das funções agregadas permitidas e $var_i$ é uma variável ligada a um atributo no corpo da regra.

O operador agregado é aplicado a um conjunto de tuplas que deve estar totalmente computado no momento da aplicação da função agregada. Para garantir isto, as regras com funções agregadas devem ser estratificadas, ou seja, as funções agregadas não podem estar envolvidas em ciclos de recursão. Adicionalmente, a variável envolvida na função deve estar ligada a um atributo no corpo da regra.

O exemplo 3.5 apresenta uma regra que utiliza uma função agregada.

**Exemplo 3.5:**

*NumeroDePacientesPorSetor(Setor:X,Pacientes:count(Y)):-*
*Pacientes(Nome:Y,Setor:X).*

Nesta regra, a relação derivada *NumeroDePacientesPorSetor* conterá o número de pacientes que estão internados em cada setor.

O subconjunto a que será aplicado o operador agregado é computado em função dos demais atributos presentes na cabeça da regra. No exemplo 3.5, a presença do atributo *Setor* na cabeça da regra, implica na aplicação do operador *count* sobre cada um dos subconjuntos formados pelo agrupamento de tuplas com o mesmo valor para o atributo *Setor*. Se na cabeça da regra aparece apenas a função, ou seja, nenhum outro atributo é mencionado, o operador agregado é aplicado sobre toda a relação, como demonstra o exemplo 3.6.

**Exemplo 3.6:**

*MediaDeIdade(Media:avg(X)) :- pacientes(Idade:X).*

Neste exemplo, a relação derivada *MediaDeIdade* conterá a média de idade de todos os pacientes da relação *pacientes*.

## 3.6 Introduzindo disjunção

Em DEDALO, disjunções são permitidas no corpo das regras. Uma regra contendo disjunção teria a forma:

$$p(atr_1{:}argc_1,atr_2{:}argc_2,...,atr_n{:}argc_n) \text{ :- } q_1(atr_1{:}arg_1,...,atr_n{:}arg_n) \text{ ;}$$
$$q_2(atr_1{:}arg_1,...,atr_n{:}arg_n),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n).$$

Seguindo a notação utilizada pelo Prolog, um ponto-e-vírgula (*;*) representa a disjunção.

A disjunção é interpretada como a união dos conjuntos das tuplas computadas por cada parte sem disjunção do corpo da regra. O exemplo 3.7 ilustra a utilização da disjunção em uma regra da linguagem DEDALO.

**Exemplo 3.7:**

*PacientesNaUTI(Nome:X) :- pacientes(Nome:X,setor:"UTI1") ;*
*pacientes(Nome:X,setor:"UTI2").*

A relação derivada *PacientesNaUTI* conterá todos os pacientes que estiverem internados no setor *"UTI1"* ou no setor *"UTI2"*. A mesma relação derivada poderia ser obtida utilizando duas regras sem disjunção, como mostra o exemplo 3.8.

**Exemplo 3.8:**

*PacientesNaUTI(Nome:X) :- pacientes(Nome:X,setor:"UTI1").*
*PacientesNaUTI(Nome:X) :- pacientes(Nome:X,setor:"UTI2").*

## 3.7 Especificando ordenação das tuplas derivadas

Em aplicações do mundo real, freqüentemente é de grande utilidade a capacidade de se especificar a ordem das tuplas contidas em uma relação derivada. Em Datalog, no entanto, a ordem das tuplas dentro da relação deduzida é determinada pela ordem em que foram geradas.

Para especificar ordenação das tuplas nas relações derivadas, a linguagem DEDALO permite que se introduza anotações na cabeça das regras que informam que atributos devem ser considerados para ordenar as tuplas derivadas. Uma regra contendo anotações na cabeça teria a forma:

$$p(atr_1{:}argc_1@n_1,atr_2{:}argc_2@n_2,...,atr_n{:}argc_n){:-}\ q_1(atr_1{:}arg_1,...,atr_n{:}arg_n\ ,$$
$$q_2(atr_1{:}arg_1,...,atr_n{:}arg_n),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n).$$

onde o símbolo @ indica a presença de uma anotação e cada um dos $n_i s$ é um número inteiro que indica a ordem das tuplas, sendo que as tuplas serão ordenadas em ordem crescente pelos valores dos atributos começando pelo menor dos $n_i s$.

O exemplo 3.9 apresenta uma regra que utiliza anotações na cabeça para determinar uma ordenação das tuplas geradas.

**Exemplo 3.9:**

*PacientesMenores(Nome:X@2,Idade:Y@1) :- Pacientes(Nome:X,Idade:Y),*
*Y < 18.*

Neste exemplo, a relação *PacientesMenores* conterá todos os pacientes que possuem idade menor que 18 anos, sendo que as tuplas geradas serão ordenadas em ordem crescente primeiro pelo atributo *idade* do paciente (marcado com *@1*) e depois em ordem crescente pelo atributo *nome* do paciente (marcado com *@2*).

Quando duas ou mais regras definem a mesma relação derivada, ou seja, possuem a mesma cabeça, a compatibilidade das cabeças deve ser observada também em relação às anotações. Se duas regras definindo a mesma relação apresentam ordenações conflitantes, um erro é gerado.

## 3.8 Introduzindo raciocínio aproximado

Aplicações tipicamente tratadas por sistemas especialistas, como a descrita no capítulo 9, freqüentemente exigem capacidade de raciocínio aproximado. Com o objetivo de introduzir a capacidade de realizar deduções baseadas neste tipo de raciocínio, foram utilizadas adaptações da lógica fuzzy para Bancos de Dados Dedutivos no sistema DEDALO.

A lógica fuzzy e os conjuntos fuzzy [ZAD 65, VIO 93] foram introduzidos na década de sessenta como uma alternativa à lógica bi-valorada, onde os fatos são totalmente verdadeiros ou totalmente falsos. Na lógica fuzzy, os fatos podem ser de 0 a 100% verdadeiros. Em um *conjunto fuzzy* a pertinência de um elemento $x$ em um conjunto $C$ é indicada por um valor entre 0 e 1, onde 0 indica certeza absoluta de que $x$ ? $C$ é falso e 1 indica certeza absoluta de que $x$ ? $C$ é verdadeiro.

Adaptações da lógica fuzzy para Bancos de Dados Dedutivos foram propostas por [MAR 90] para o sistema LogiQuel [MAR 91]. Esta proposta utiliza a noção de conjuntos fuzzy para determinar se uma tupla pertence ou não à relação derivada. Na linguagem e no sistema DEDALO, o raciocínio aproximado é tratado de forma similar ao sistema LogiQuel.

Cada relação base do Banco de Dados deve possuir um atributo com um nome padronizado *cf* (*confidence factor*), cujo valor varia de 0 a 1 e indica o fator de certeza da tupla. A linguagem DEDALO permite que se associe a cada regra um fator de certeza que determina a confiança na verdade da regra. Uma regra com fator de certeza teria a forma:

$$p(atr_1{:}argc_1,atr_2{:}argc_2,...,atr_n{:}argc_n){:-}\ q_1(atr_1{:}arg_1,...,atr_n{:}arg_n)\ ,\ q_2(atr_1{:}arg_1,...,atr_n{:}arg_n),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n)\ cf\ v.$$

onde *cf* indica que a regra possui um fator de certeza, representado por *v* que é um número cujo valor é maior que 0 e menor ou igual a 1.

Cada tupla derivada deve ter também um fator de certeza. Este fator é computado a partir dos fatores de certeza das tuplas das relações do corpo da regra e do fator de certeza da regra. O fator das tuplas derivadas é obtido pela multiplicação do fator de certeza computado do corpo da regra pelo fator de certeza da própria regra. A computação do fator de certeza do corpo da regra é feito a partir da seguinte adaptação das leis da teoria dos conjuntos fuzzy:

Sendo $q_i$ um dos literais do corpo da regra e $?_i$ o conjunto de argumentos do literal $q_i$:

$CF(q_1(?_1)\ ?\ q_2(?_2)) = \max\{\ CF(q_1(?_1))\ ,\ CF(q_2(?_2))\ \}$

$CF(q_1(?_1)\ ?\ q_2(?_2)) = \min\{\ CF(q_1(?_1))\ ,\ CF(q_2(?_2))\ \}$

$CF(?\ q_1(?_1)) = 1 - CF(q_1(?_2))$

Intuitivamente, isto significa que, na presença de disjunções, o fator de certeza é o maior entre os fatores das tuplas envolvidas na disjunção, para conjunções o fator é o menor entre os fatores e, na presença de negação, o fator da tupla negada é o resultado da subtração 1 menos o fator de certeza da tupla.

Sendo *P* a cabeça da regra e *Q* o corpo, o fator de certeza de uma relação derivada computada a partir da regra *P :- Q*, é obtido da seguinte forma:

$CF(P) = CF(Q) * CF(P :- Q)$

Por fim, para fatos derivados com fatores de certeza diferentes ("certeza multiplamente argüida"), é aplicada a lei da disjunção para manter na relação derivada apenas os fatos que possuem o maior fator de certeza. Sendo que $f_1$ e $f_2$ representam a mesma tupla *f*, mas foram derivadas com fatores de certeza diferentes, o fator de certeza da tupla *f* é:

$CF(f) = \max\{CF(f_1)\ ,\ CF(f_2)\ \}$

Tuplas que possuem fator de certeza zero não são mantidas na relação, pois o fator de certeza zero indica certeza absoluta que a tupla é falsa e, pela Hipótese do Mundo Fechado, toda tupla que não está presente na relação é assumida falsa.

O exemplo 3.10 demonstra utilização de uma regra com fator de certeza.

**Exemplo 3.10:**

*Crianças(Nome:X) :- pacientes(Nome:X,Idade:Y), Y < 14 cf 0,9.*

A regra do exemplo 3.10 indica que os pacientes com menos de 14 anos são considerados crianças com 90% de certeza.

## 3.9 Introduzindo operações de atualização

Como foi idealizado para ser uma linguagem de consulta, o Datalog não permite que se expressem operações de atualização utilizando sua linguagem. No entanto, a capacidade de expressar atualizações usando a mesma linguagem utilizada nas consultas é altamente desejável.

As operações de atualização (inserção, exclusão e modificação) podem ser solicitadas na linguagem DEDALO utilizando um dos operadores de atualização providos pela linguagem. Estes operadores são: *ins*, para inserções, *del* para exclusões e *upd* para modificações.

A forma de utilização de cada um dos operadores de atualização é muito semelhante. Basicamente, cada um deles pode ser aplicado diretamente sobre uma relação ou em uma regra de atualização. A aplicação do operador diretamente sobre uma relação básica ou derivada, indica a solicitação da operação de atualização incondicionalmente. Caso o operador seja aplicado sobre uma regra, cuja cabeça é uma relação básica ou derivada em que se deseja efetuar a atualização, a operação será restrita às tuplas que satisfazem as condições expressas no corpo da regra.

Desta forma, uma operação de inserção pode ser solicitada com uma das seguintes sintaxes:

Sintaxe I.1: $ins\ p(atr_1{:}c_1,atr_2{:}c_2,...,atr_n{:}c_n).$

onde $p$ é uma relação básica ou derivada em que se deseja inserir novas tuplas, $atr_1...atr_n$ são nomes de atributos da relação e $c_1...c_n$ são constantes.

Sintaxe I.2: $ins\ p(atr_1{:}argc_1,...,atr_n{:}argc_n){:}\text{-}$
$q_1(atr_1{:}arg_1,...,atr_n{:}arg_n)\ ,...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n),\ C,\ cf\ v.$

onde $p$ é uma relação básica ou derivada em que se deseja inserir novas tuplas, $q_1...q_n$ é o corpo da regra de inserção, sendo que cada um dos $q_i$s representa uma relação básica ou derivada, $atr_1...atr_n$ são nomes de atributos da relação, $argc_1...argc_n$ são variáveis ligadas a atributos do corpo, ou constantes, ou operações aritméticas, ou funções agregadas, $arg_1...arg_n$ são variáveis ligadas a atributos, ou constantes, ou operações aritméticas, $C$ é uma operação de comparação opcional, $cf$ indica que a regra possui um fator de certeza, representado por $v$ que é um número cujo valor é maior que 0 e menor ou igual a 1.

A solicitação de inserção feita com a sintaxe I.1 inseriria uma nova tupla na relação $p$ cujos valores dos atributos seriam aqueles representados pelas constantes $c_i$. Obviamente, neste tipo de solicitação de inserção, não podem existir variáveis associadas aos atributos e para todos os atributos cujo valor não pode ser deixado como nulo, um valor constante deve ser informado. O fator de certeza da tupla inserida é 1 por default. Para se especificar um fator diferente, um dos $atr_i$ deve ser $cf$. O exemplo 3.11 demonstra a utilização desta sintaxe na especificação de inserções.

**Exemplo 3.11:**

*ins pacientes(nome:'joão', idade:25, setor:'UTI').*

Neste exemplo, uma nova tupla seria inserida na relação *pacientes*, onde os atributos *nome*, *idade* e *setor* teriam respectivamente os valores *'joão'*, *25* e *'UTI'*.

A solicitação de inserção feita com a sintaxe I.2, inseriria na relação *p* as tuplas derivadas da aplicação do corpo da regra sobre o Banco de Dados. Da mesma forma que com a sintaxe I.1, todos os atributos cujo valor não pode ser deixado como nulo deverão possuir um valor associado, que pode ser uma constante ou mesmo uma variável, ou uma função agregada ou uma operação aritmética definida em termos dos atributos das relações do corpo da regra. O fator de certeza das novas tuplas é computado em função do corpo da regra, de forma semelhante ao que é feito nas regras de derivação, a menos que um fator de certeza seja explicitamente declarado, especificando *cf* como um dos $atr_i$. O exemplo 3.12 demonstra a utilização desta sintaxe na inserção de novas tuplas.

**Exemplo 3.12:**

*ins crianças(nome:x,idade:y) :- pacientes(nome:x,idade:y),y < 14.*

Neste exemplo, seriam inseridos na relação *crianças* todos os pacientes com idade menor que 14 anos.

Uma operação de modificação poderia ser solicitada com uma das seguintes sintaxes:

Sintaxe U.1: $upd\ p(atr_1{:}c_1,atr_2{:}c_2,...,atr_n{:}c_n).$

onde *p* é uma relação básica ou derivada em que se deseja modificar tuplas, $atr_1...atr_n$ são nomes de atributos da relação e $c_1...c_n$ são constantes.

Sintaxe U.2: $upd\ p(atr_1{:}argc_1,...,atr_n{:}argc_n){:}-$
$q_1(atr_1{:}arg_1,...,atr_n{:}arg_n)\ ,...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n),\ C.$

onde *p* é uma relação básica ou derivada em que se deseja modificar tuplas, $q_1...q_n$ é o corpo da regra de modificação, sendo que cada um dos $q_i s$ representa uma relação básica ou derivada, $atr_1...atr_n$ são nomes de atributos da relação, $argc_1...argc_n$ são variáveis ligadas a atributos do corpo, ou constantes, ou operações aritméticas, ou funções agregadas, $arg_1...arg_n$ são variáveis ligadas a atributos, ou constantes, ou operações aritméticas e *C* é uma operação de comparação opcional.

A solicitação de modificação feita com a sintaxe U.1 alteraria os valores dos atributos especificados pelos $atr_i$ de todas as tuplas da relação *p* para os valores das constantes $c_i$. Obviamente, neste tipo de solicitação de modificação, não podem existir variáveis associadas aos atributos da relação *p*. O exemplo 3.13 demonstra a utilização desta sintaxe na alteração de tuplas.

**Exemplo 3.13:**

*upd pacientes(idade:20).*

A operação do exemplo 3.13 alteraria em todas as tuplas da relação *pacientes* o valor do atributo *idade* para *20*.

A solicitação de modificação feita com a sintaxe U.2, modificaria na relação *p* as tuplas que satisfazem as condições expressas no corpo da regra, como demonstrado pelo exemplo 3.14.

**Exemplo 3.14:**

*upd pacientes(conta: x + x * 0.10) :- pacientes(setor:z,conta:x) , z = 'UTI'.*

A regra de modificação do exemplo 3.14 atualiza os valores do atributo *conta* da relação *pacientes* em 10% para todas as tuplas cujo valor do atributo *setor* seja *'UTI'*.

Uma operação de exclusão poderia ser solicitada com uma das seguintes sintaxes:

Sintaxe D.1: *del p.*

onde *p* é uma relação básica ou derivada de que se deseja excluir tuplas.

Sintaxe D.2: *del* $p(atr_1{:}c_1,atr_2{:}c_2,...,atr_n{:}c_n)$.

onde *p* é uma relação básica ou derivada de que se deseja excluir tuplas, $atr_1...atr_n$ são nomes de atributos da relação e $c_1...c_n$ são constantes.

Sintaxe D.3: *del* $p(atr_1{:}argc_1,...,atr_n{:}argc_n)$ :-
$q_1(atr_1{:}arg_1,...,atr_n{:}arg_n),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n)$, *C.*

onde *p* é uma relação básica ou derivada de que se deseja excluir tuplas, $q_1...q_n$ é o corpo da regra de exclusão, sendo que cada um dos $q_i$*s* representa uma relação básica ou derivada, $atr_1...atr_n$ são nomes de atributos da relação, $argc_1...argc_n$ são variáveis ligadas a atributos do corpo, ou constantes, ou operações aritméticas, ou funções agregadas, $arg_1...arg_n$ são variáveis ligadas a atributos, ou constantes, ou operações aritméticas e *C* é uma operação de comparação opcional.

A solicitação de exclusão feita com a sintaxe D.1 excluiria incondicionalmente todas as tuplas da relação *p*, como demonstrado pelo exemplo 3.15.

**Exemplo 3.15:**

*del pacientes.*

Esta operação excluiria todas as tuplas da relação *pacientes*.

A solicitação de exclusão feita com a sintaxe D.2, excluiria da relação *p* as tuplas cujos valores de atributos fossem iguais às constantes $c_i$, como demonstra o exemplo 3.16.

**Exemplo 3.16:**

*del pacientes(setor:'UTI').*

Neste exemplo, todas as tuplas da relação *pacientes* cujo valor do atributo *setor* é *'UTI'* seriam excluídas.

A solicitação de exclusão feita com a sintaxe D.3, excluiria da relação *p* as tuplas que pudessem ser derivadas da aplicação da regra sobre o Banco de Dados, como demonstra o exemplo 3.17.

**Exemplo 3.17:**

*del pacientes(Nome:X) :- pacientes(Nome:X,Idade:Y), Y < 15.*

Neste exemplo, todas as tuplas da relação *pacientes* cujo valor do atributo *idade* é menor que *15* seriam excluídas.

## 3.10 Garantindo a segurança das regras

Um conceito de grande importância em Bancos de Dados Dedutivos é a noção de *regras seguras* ou *permitidas*. A segurança de uma regra é garantida por uma verificação puramente sintática que decide se uma regra é tratável ou não. Uma regra é dita tratável se na sua avaliação todas as variáveis estarão instanciadas com constantes. Este conceito é essencial para garantir a computabilidade da negação, operações aritméticas, operações de comparação e funções agregadas.

Em [CER 90], uma regra é considerada segura se e somente se todas as variáveis que ocorrem em um literal negativo ou em uma operação de comparação ocorrem também em um literal positivo. Isto garante que no momento da avaliação da regra, as variáveis do literal negativo ou as da operação de comparação estarão instanciadas. O exemplo 3.18 demonstra o que aconteceria na avaliação de uma regra não segura.

**Exemplo 3.18:**

*sadios(nome:x) :- not(pacientes(nome:x)).*

A relação *sadios* deveria armazenar todas as pessoas que não estão na relação pacientes. No entanto, com a adoção da Hipótese do Mundo Fechado para computar a negação, somente os fatos positivos são conhecidos. Os fatos negativos não são armazenados no Banco de Dados. Desta forma, é impossível derivar a relação *sadios*. A mesma relação poderia ser computada de forma segura com a regra do exemplo 3.19.

**Exemplo 3.19:**

*sadios(nome:x) :- pessoas(nome:x) , not(pacientes(nome:x)).*

Com esta regra, a relação *sadios* conteria todas as tuplas da relação *pessoas* que não possuem uma tupla correspondente na relação *pacientes*. A regra é segura pela definição de [CER 90].

Da mesma forma que na negação, a correta avaliação de operações aritméticas, operações de comparação e funções agregadas, claramente dependem de que as variáveis estejam instanciadas no momento de sua avaliação. Para garantir a computabilidade das regras, a seguinte noção de regras seguras, um pouco diferente da noção encontrada em [CER 90], é utilizada na linguagem DEDALO.

**Definição 3.1: Regras seguras:**

Uma regra é segura se e somente se, para cada parte sem disjunção do corpo da regra:

1. Pelo menos uma variável de cada literal negativo ocorra em um literal positivo do corpo da regra ou pelo menos um atributo de cada literal negativo esteja ligado a uma constante.

2. Todas as variáveis envolvidas em operações aritméticas, operações de comparação e funções agregadas ocorram em um literal positivo do corpo da regra ou em um literal negativo que já satisfaça a condição 1.

A condição 1 exige que apenas um dos atributos do literal negativo esteja ligado a constantes, diretamente ou através de uma variável presente em um literal positivo, e isto é suficiente para garantir a computabilidade da regra, não sendo necessário que todas as variáveis presentes em literais negativos estejam presentes também em literais positivos como exige a definição encontrada em [CER 90]. A razão para isto é que tendo-se uma condição no literal negado, ou seja, a presença de um constante ou de uma variável presente em um literal positivo, pode-se ver as demais variáveis como valores sem relevância e computar com segurança a negação, como demonstra o exemplo 3.20.

**Exemplo 3.20:**

*sadios(nome:x) :- pessoas(nome:x) , not(pacientes(nome:x,idade:y)).*

Mesmo que a variável *y* do literal negativo *pacientes* não apareça em nenhum literal positivo, a negação pode ser computada com segurança. A relação *sadios* conterá todas as pessoas que não tenham uma tupla correspondente na relação *pacientes*, não importando qual o valor do atributo *idade*.

A condição de segurança é verificada para cada parte sem disjunção da regra porque uma regra com disjunção é vista como duas regras com a mesma cabeça, ou seja, uma regra com disjunção:

*p(at1:v1,at2:v2) :- q1(at1:v1,at2:v2) ; q2(at1:v1,at2:v2).*

é o mesmo que:

*p(at1:v1,at2:v2) :- q1(at1:v1,at2:v2).*

e

*p(at1:v1,at2:v2) :- q2(at1:v1,at2:v2).*

Além da exigência de que as regras sejam seguras, para garantir que todos os atributos das relações derivadas possuam valores, cada variável que aparece na cabeça da regra deve aparecer também no corpo da regra.

## 3.11 Comentários finais

A linguagem DEDALO busca suprir as deficiências do Datalog puro através da introdução de uma série de extensões. O objetivo principal é obter uma linguagem declarativa que possa ser utilizada com eficiência em situações práticas e que tenha poder expressivo maior que a álgebra relacional. A aplicação descrita no capítulo 9, que reúne características de aplicações tradicionais de Bancos de Dados e características de aplicações típicas de sistemas especialistas, serviu como base para testar a aplicabilidade das extensões introduzidas pela linguagem DEDALO, utilizada como linguagem de consulta e atualização pela aplicação, obtendo-se resultados satisfatórios no que se refere aos objetivos propostos.

# 4 Gerenciando a propagação de atualizações em Sistemas de Bancos de Dados Dedutivos

Em um sistema de Banco de Dados Dedutivo, o conteúdo das relações derivadas depende do conteúdo das relações básicas do Banco de Dados. Assim, o efeito de uma atualização sobre uma relação básica deve ser visível nas relações derivadas. Da mesma forma, se é solicitada uma atualização sobre uma relação derivada, as relações básicas devem ser modificadas de forma que a atualização solicitada se torne visível.

Pode-se utilizar duas políticas básicas para tratar relações derivadas. A primeira é manter as relações derivadas *virtuais*, ou seja, não são armazenadas e são computadas quando necessário. A segunda política é armazenar as relações derivadas, ou seja, manter as relações derivadas *materializadas*. A vantagem das relações derivadas materializadas é que se pode diminuir o tempo de acesso, através da criação de estruturas de índice. No entanto, se uma modificação nas relações básicas causa uma alteração nas relações derivadas materializadas, estas devem ser modificadas de forma a refletir a modificação realizada nas relações básicas. Este processo é chamado de *propagação de atualizações.*

Na maioria das vezes, não é a melhor solução recomputar toda a relação derivada materializada a cada modificação do Banco de Dados. Esta alternativa seria interessante nos casos em que ocorre uma grande quantidade de modificações no BD, como a exclusão de uma relação inteira, por exemplo. Como esta situação não é a mais comum, a melhor solução é a utilização de *algoritmos incrementais* que realizam nas relações derivadas apenas as alterações correspondentes às modificações nas relações básicas.

O problema da propagação de atualizações sobre relações básicas para relações derivadas materializadas e as suas soluções, são os assuntos deste capítulo.

Nos últimos anos, várias técnicas de propagação de atualizações para relações derivadas materializadas foram propostas. Algumas são aplicáveis a qualquer tipo de relação derivada, outras a relações derivadas definidas por regras com algumas características especiais, como sem negação e não recursivas, por exemplo. Como estas últimas são eficientes na manutenção de suas classes de relações derivadas, na implementação do Sistema DEDALO, optou-se por escolher uma técnica para cada tipo de relação derivada. Assim, as relações derivadas foram classificadas de acordo com suas regras de formação da seguinte forma:

1. Relações derivadas definidas por uma única regra sem negação

2. Relações derivadas definidas por múltiplas regras sem recursão

3. Relações derivadas definidas por regras recursivas

4. Relações derivadas definidas por regras com negação

A partir desta classificação, várias técnicas de propagação de atualizações foram estudadas e para cada uma das classes uma técnica foi escolhida. Uma breve introdução às técnicas estudadas e a descrição da técnica adotada pelo Sistema DEDALO para cada uma das classes são apresentadas a seguir.

## 4.1 Propagação para relações derivadas definidas por uma única regra sem negação

Relações derivadas definidas por uma única regra não podem ser recursivas, pois a criação de uma relação derivada recursiva não vazia exige pelo menos duas regras. Para este tipo de relação derivada, foi proposto em [PAR 94] uma classificação dos literais do corpo da regra, que é utilizada na escolha do método de propagação. A classificação proposta é a seguinte:

? **Literais restritos**: um literal é restrito se pelo menos a chave da relação a que ele corresponde é composta de atributos restritos. Os atributos restritos de uma regra são obtidos pela aplicação do seguinte algoritmo:

**Algoritmo 1:**

1.1. Os atributos da relação na cabeça da regra são inseridos no conjunto de atributos restritos;

1.2. Adiciona-se ao conjunto de atributos restritos:

(a) Todos os atributos que aparecem em uma condição de igualdade com um valor constante;

(b) Todos os atributos que aparecem em uma condição de igualdade com um atributo já no conjunto de atributos restritos;

1.3. Repete-se o passo 1.2 até que um ponto fixo seja atingido.

O exemplo 4.1 demonstra a utilização do algoritmo 1 na obtenção do conjunto de literais restritos.

**Exemplo 4.1:** Considerando que o atributo *Nome* é chave da relação *pacientes:*

*pacientes_na_UTI(Nome:X) :- pacientes(Nome:X,Setor:'UTI1').*

Algoritmo 1:

Passo 1.1: Atributos restritos: *Nome*

Passo 1.2: Atributos restritos: *Nome, Setor*

Como *Nome* é a chave para *pacientes* e *Nome* está no conjunto dos atributos restritos, o literal *pacientes* é restrito.

Os literais restritos possuem duas propriedades que são utilizadas na escolha do método de propagação:

1. Cada tupla da relação derivada corresponde a apenas uma tupla da relação correspondente ao literal restrito.

2. Dada uma tupla da relação correspondente ao literal restrito, é possível identificar as tuplas da relação derivada que são derivadas dela sem acessar outras relações.

? **Literais seguros:** Um literal é seguro se todos os atributos da relação a que ele corresponde estão no conjunto de atributos cobertos. O conjunto de atributos cobertos de uma regra é obtido pela aplicação do seguinte algoritmo:

**Algoritmo 2:**

2.1. Os atributos da relação na cabeça da regra são inseridos no conjunto de atributos cobertos;

2.2. Adiciona-se ao conjunto de atributos cobertos todos os atributos que aparecem em uma condição de igualdade com um valor constante;

2.3. Repete os passos (a) e (b) até que um ponto fixo seja atingido:

(a) Adiciona-se ao conjunto dos atributos cobertos todos os atributos que aparecem em uma condição de igualdade com um atributo já no conjunto de atributos cobertos;

(b) Para cada relação, se todos os atributos que fazem parte da chave estão no conjunto dos atributos cobertos, adicionar também todos os demais atributos da relação.

O exemplo 4.2 demonstra a utilização do algoritmo 2 na obtenção do conjunto de literais cobertos.

**Exemplo 4.2:** Considerando que *Nome* é chave de *pacientes* e *Código* é chave para *setores*:

*Pacientes_na_UTI(Nome:X) :- pacientes(Nome:X,Setor:Y) ,*
*Setores(Código:Y,Descrição:'UTI').*

Algoritmo 2:

Passo 2.1: Atributos cobertos: *Nome*

Passo 2.2: Atributos cobertos: *Nome,Descrição*

Passo 2.3.b: Atributos cobertos: *Nome,Descrição, Setor*

Passo 2.3.a: Atributos cobertos: *Nome, Descrição, Setor, Código*

Como todos os atributos das relações *pacientes* e *setores* estão no conjunto de atributos cobertos, ambos os literais são seguros.

Os literais restritos são, na verdade, um subconjunto dos literais seguros. Todo literal restrito é também seguro. Como os literais restritos, os literais seguros possuem a seguinte propriedade:

? Cada tupla da relação derivada corresponde a apenas uma tupla da relação correspondente ao literal seguro.

Ao contrário do que acontece com os literais restritos, dada uma tupla da relação correspondente ao literal seguro, não é possível identificar as tuplas da relação derivada que são derivadas dela sem acessar outras relações.

? **Literais não seguros:** Se um literal não é restrito nem seguro, ele é não seguro.

O exemplo 4.3 apresenta uma regra com um literal não seguro:

**Exemplo 4.3:** Considerando que *Código* é a chave para *setores*:

*Setor_lotado(Descrição:X,No_pacientes:Y) :-*
*Setores(Código:Z,Descrição:X, No_pacientes:Y) , Y > 100.*

Atributos restritos: *Descrição, No_pacientes*

Atributos cobertos: *Descrição, No_pacientes*

O literal *setores* não é restrito porque sua chave não se encontra no conjunto de atributos restritos e não é seguro porque nem todos os seus atributos se encontram no conjunto de atributos cobertos. Portanto, o literal *setores* é não seguro.

A partir desta classificação, é possível adotar métodos de acordo com a classe dos literais. Existem métodos específicos para cada tipo de literal. Para os literais restritos, for criado o *método de restrição* [PAR 94], para os literais seguros, o *método da chave* [BLA 86, CER 91] e para os literais não seguros, os métodos *DRed* [CER 94, GUP 93] e *Counting* [BLA 86, GUP 92]. Cada um destes métodos será descrito a seguir.

### 4.1.1 Método de restrição

O método de restrição [PAR 94] deriva modificações para uma visão materializada a partir das modificações detectadas nas relações representadas por literais restritos. Este método insere uma nova tupla na relação derivada materializada se foi inserida uma tupla na relação correspondente ao literal restrito e se existe uma substituição válida para os demais literais do corpo da regra. O método de restrição faz uso das propriedades dos literais restritos para determinar que tuplas devem ser excluídas de uma relação derivada materializada, em função das tuplas que foram excluídas da relação correspondente ao literal restrito.

Este método é implementado derivando-se para cada literal restrito de uma regra uma solicitação de inserção e uma solicitação de exclusão. Uma modificação é tratada como uma exclusão seguida da inserção da tupla modificada. Para uma regra

contendo um literal restrito, as solicitações de inserção e exclusão derivadas, seguindo a sintaxe da linguagem DEDALO seriam as seguintes:

Regra contendo um literal restrito:

$$p(?_d) :- q_1(?_1) ,...,q_i(?_2),...,q_n(?_n), C\ cf\ v.$$

onde $p$ é a relação derivada materializada, $q_i$ é o literal restrito, $?_i$ é o conjunto de atributos de cada relação, $C$ são as operações de comparação da regra e $v$ é o valor do fator de certeza da regra.

Solicitação de inserção:

$$ins\ p(?_d) :- q_1(?_1) ,...,ins\_q_i(?_i),...,q_n(?_n), C\ cf\ v.$$

onde $ins\_q_i$ é uma relação delta de inserção. As relações *delta* [CHA 93, DAY 94, SIM 92] são relações definidas pelo sistema que armazenam as tuplas inseridas e excluídas durante a transação. Assim, se $q(A1,...,An)$ é o esquema de uma relação, então existirão associadas a $q$ as seguintes relações delta:

$?\ ins\_q(A1,...,An)$

$?\ del\_q(A1,...,An)$

Intuitivamente, $ins\_q(A1,...,An)$ armazena as tuplas inseridas em $q$ durante a transação, e $del\_q(A1,...,An)$ armazena as tuplas excluídas de $q$ durante a transação.

Solicitação de exclusão:

$$del\ p(?_{di}) :- p(?_{di}), del\_q_i(?_i), C_i.$$

onde $?_{di}$ representa os atributos de $p$ que aparecem também no literal restrito $q_i$, $C_i$ representa as operações de comparação da regra em que estão envolvidos constantes e atributos de $q_i$ e $del\_q_i$ é uma relação delta que armazena as tuplas excluídas da relação $q_i$ durante a transação.

A solicitação de inserção irá inserir uma tupla na relação $p$ se foi inserida uma tupla na relação $q_i$, existe uma substituição válida para os demais literais do corpo e as operações de comparação $C$ são verdadeiras. O fator de certeza das tuplas inseridas é calculado em função do corpo e do fator de certeza da regra. Se a tupla a ser inserida já se encontra na relação $p$, a ação é nula.

A solicitação de exclusão irá excluir da relação $p$ as tuplas correspondentes às tuplas excluídas do literal restrito. A propriedade dos literais restritos de que não é necessário acessar outras relações para determinar quais as tuplas da relação derivada que correspondem a determinada tupla da relação correspondente ao literal restrito, é utilizada para derivar a solicitação de exclusão, pois apenas a relação correspondente ao literal restrito e a relação derivada materializada são acessadas.

A solicitação de exclusão deve ser executada antes da solicitação de inserção para garantir a correção do método. Também deve ser observado que se uma tupla é inserida durante uma transação e posteriormente, durante a mesma transação, é

excluída, a tupla deve ser excluída também da relação que armazena as tuplas inseridas. O exemplo 4.4 demonstra a derivação das solicitações de atualização para a regra do exemplo 4.1.

**Exemplo 4.4:** Considerando a seguinte regra, cujo literal *pacientes* é restrito:

*pacientes_na_UTI(Nome:X) :- pacientes(Nome:X,Setor:'UTI1').*

As seguintes solicitações de atualização seriam derivadas pelo método de restrição:

*ins pacientes_na_UTI(Nome:X) :- ins_pacientes(Nome:X,Setor:'UTI1').*
*del pacientes_na_UTI(Nome:X) :- pacientes_na_UTI(Nome:X),*
*del_pacientes(Nome:X,Setor:'UTI1').*

A solicitação de inserção irá inserir na relação *pacientes_na_UTI* todos os pacientes que foram inseridos durante a transação, e portanto estão na relação delta *ins_pacientes*, cujo setor é 'UTI1'.

A solicitação de exclusão irá excluir da relação *pacientes_na_UTI* todos os pacientes que foram excluídos durante a transação, e portanto estão na relação delta *del_pacientes*, cujo setor era 'UTI1'.

Uma otimização adicional para este método seria modificar a ordem dos literais nas solicitações de inserção e exclusão, de forma que o literal correspondente às relações delta de inserção e exclusão do literal restrito seja o primeiro literal no corpo da regra. A razão para isto é que, na maioria das vezes, a relação delta é a menor entre as relações do corpo.

### 4.1.2 Método da Chave

O método da chave [BLA 86, CER 91] deriva modificações para uma visão materializada a partir das modificações detectadas nas relações representadas por literais seguros. Este método insere uma nova tupla na relação derivada materializada se foi inserida uma tupla na relação correspondente ao literal seguro e se existe uma substituição válida para os demais literais do corpo da regra. O método da chave exclui uma tupla da relação derivada materializada se foi excluída uma tupla da relação correspondente ao literal seguro e se, no último estado válido do BD antes da transação, havia uma substituição válida para os demais literais do corpo da regra.

Da mesma forma que o método de restrição, este método é implementado derivando-se para cada literal seguro de uma regra uma solicitação de inserção e uma solicitação de exclusão. Uma modificação é tratada como uma exclusão seguida da inserção da tupla modificada. Para uma regra contendo um literal seguro, as solicitações de inserção e exclusão derivadas, seguindo a sintaxe da linguagem DEDALO seriam as seguintes:

Regra contendo um literal seguro:

$p(?_d)$ :- $q_1(?_1)$ ,...,$q_i(?_i)$,...,$q_n(?_n)$, *C cf v.*

onde $p$ é a relação derivada materializada, $q_i$ é o literal seguro, ? é o conjunto de atributos de cada relação, $C$ são as operações de comparação da regra e $v$ é o valor do fator de certeza da regra.

Solicitação de inserção:

$$ins\ p(?) :- q_1(?_1), ..., ins\_q_i(?_i), ..., q_n(?_n), C\ cf\ v.$$

onde $ins\_q_i$ é uma relação delta que armazena as inserções feitas sobre a relação $q_i$ durante a transação.

Solicitação de exclusão:

$$del\ p(?_d) :- p(?_d), old\_q_1(?_1), ..., del\_q_i(?_i) ..., old\_q_n(?_n), C.$$

onde $del\_q_i$ é uma relação delta que armazena as tuplas excluídas da relação $q_i$ durante a transação e $old\_q$ representa o conteúdo da relação $q$ no último estado válido antes da transação.

O conteúdo de uma relação no último estado válido do BD pode ser obtido pela seguinte equação:

$$(R - ins\_R) ?\ \ del\_R$$

onde $R$ é o conteúdo da relação no estado atual do BD, $ins\_R$ é a relação delta que armazena as tuplas inseridas em $R$ durante a transação e $del\_R$ é a relação delta que armazena as tuplas que foram excluídas de $R$ durante a transação.

Aplicando esta equação, a solicitação de exclusão derivada pelo método da chave seria:

$$del\ p(?_d) :- p(?_d), ((q_1(?_1), not(ins\_q_1(?_1))) ; del\_q_1(?_1)), ...,$$
$$del\_q_i(?_i), ..., ((q_n(?_n), not(ins\_q_n(?_n))) ; del\_q_n(?_n)), C.$$

A solicitação de inserção irá inserir uma tupla na relação $p$ se foi inserida uma tupla na relação $q_i$, existe uma substituição válida para os demais literais do corpo e as operações de comparação $C$ são verdadeiras. O fator de certeza das tuplas inseridas é calculado em função do corpo e do fator de certeza da regra. Se a tupla a ser inserida já se encontra na relação $p$, a ação é nula.

A solicitação de exclusão irá excluir da relação $p$ as tuplas correspondentes às tuplas excluídas do literal seguro, se existia uma substituição válida para os demais literais do corpo da regra no último estado válido antes da transação e se as operações de comparação $C$ são verdadeiras.

A solicitação de exclusão deve ser executada antes da solicitação de inserção para garantir a correção do método. Também deve ser observado que se uma tupla é inserida durante uma transação e posteriormente, durante a mesma transação, é excluída, a tupla deve ser excluída também da relação que armazena as tuplas inseridas. A solicitação de inserção é idêntica à do método de restrição. A diferença entre os dois métodos está na solicitação de exclusão.

O exemplo 4.5 demonstra a derivação das solicitações de atualização para a regra do exemplo 4.2.

**Exemplo 4.5:** Considerando a seguinte regra, com os literais seguros *pacientes* e *setores*:

*Pacientes_na_UTI(Nome:X) :- pacientes(Nome:X,Setor:Y) ,*
*setores(Código:Y,Descrição:'UTI').*

As seguintes solicitações de atualização seriam derivadas pelo método da chave:

*ins pacientes_na_UTI(Nome:X) :- ins_pacientes(Nome:X,Setor:Y),*
*setores(Código:Y,Descrição:'UTI').*

*ins pacientes_na_UTI(Nome:X) :- pacientes(Nome:X,Setor:Y),*
*ins_setores(Código:Y,Descrição:'UTI').*

*del pacientes_na_UTI(Nome:X) :- pacientes_na_UTI(Nome:X),*
*del_pacientes(Nome:X,Setor:Y) ,*
*((setores(Código:Y,Descrição:'UTI') ,*
*not(ins_setores(Código:Y,Descrição:'UTI'))) ;*
*del_setores(Código:Y,Descrição:'UTI')).*

*del pacientes_na_UTI(Nome:X) :- pacientes_na_UTI(Nome:X),*
*((pacientes(Nome:X,Setor:Y) ,*
*not(ins_pacientes(Nome:X,Setor:Y))) ;*
*del_pacientes(Nome:X,Setor:Y)) ,*
*del_setores(Código:Y,Descrição:'UTI1') .*

### 4.1.3 Método DRed

O método DRed [CER 94, GUP 93] deriva modificações para uma visão materializada a partir das modificações detectadas nas relações representadas por literais não seguros. Este método deriva três solicitações de atualização: uma solicitação de exclusão, que executa todas as possíveis exclusões da relação derivada materializada em razão das exclusões ocorridas durante a transação; uma solicitação de inserção, que realiza todas as possíveis inserções na relação derivada em razão das inserções ocorridas durante a transação e uma solicitação de re-inserção, que re-insere na relação derivada materializada as tuplas que foram incorretamente excluídas na solicitação de exclusão.

Uma modificação é tratada como uma exclusão seguida da inserção da tupla modificada. Para uma regra contendo um literal não seguro, as solicitações de inserção e exclusão derivadas seriam idênticas àquelas do método de restrição ou do método da chave (ambos são adequados). Além dessas, é criada uma solicitação de re-inserção, que deve ser executada logo após a solicitação de exclusão. Seguindo a sintaxe da linguagem DEDALO a solicitação de re-inserção seria a seguinte:

Regra contendo um literal não seguro:

$p(?_d)$ :- $q_1(?_1)$ ,...,$q_i(?_i)$,...,$q_n(?_n)$, *C cf v.*

onde *p* é a relação derivada materializada, $q_i$ é o literal não seguro, ? é o conjunto de atributos de cada relação, *C* são as operações de comparação da regra e *v* é o valor do fator de certeza da regra.

Solicitação de re-inserção:

*ins* $p(?_d)$ :- *del_p*$(?_d)$ , $q_1(?_1)$ ,...,$q_i(?_i)$,...,$q_n(?_n)$, *C cf v.*

onde *del_p* é uma relação delta que armazena as tuplas excluídas da relação *p* durante a transação.

A solicitação de re-inserção irá inserir na relação derivada materializada *p* todas as tuplas que foram excluídas de *p*, mas possuíam uma derivação alternativa no estado atual do Banco de Dados.

Em [PAR 94] uma otimização adicional a este método foi proposta: se a regra possui literais seguros ou restritos, deve-se executar as solicitações de re-inserção logo após a execução das solicitações de exclusão derivadas para os literais não seguros e antes da execução das solicitações de exclusão derivadas para os literais seguros e restritos. Isto é útil porque somente a execução da solicitação de exclusão dos literais não seguros pode levar a exclusões incorretas. As solicitações de exclusão derivadas para os literais seguros e restritos sempre realizam exclusões corretas em função das propriedades destes literais.

O exemplo 4.6 demonstra a derivação das solicitações de atualização para a regra do exemplo 4.3.

**Exemplo 4.6:** Considerando a seguinte regra, com o literal não seguro *setores*:

*Setor_lotado(Descrição:X,No_pacientes:Y) :-*
*Setores(Código:Z,Descrição:X, No_pacientes:Y) ,Y > 100.*

As seguintes solicitações de atualização seriam derivadas pelo método DRed:

(exclusão) *del Setor_lotado(Descrição:X,No_pacientes:Y) :-*
*Setor_lotado(Descrição:X,No_pacientes:Y),*
*del_Setores(Código:Z,Descrição:X,No_pacientes:Y) ,Y > 100.*

(re-inserção) *ins Setor_lotado(Descrição:X, No_pacientes:Y) :-*
*del_Setor_lotado(Descrição:X,No_pacientes:Y),*
*Setores(Código:Z,Descrição:X, No_pacientes:Y),Y>100.*

(inserção) *ins Setor_lotado(Descrição:X,No_pacientes:Y) :-*
*ins_Setores(Código:Z,Descrição:X, No_pacientes:Y) ,Y > 100.*

As solicitações devem ser executadas nesta ordem. O método de restrição foi utilizado para a criação da solicitação de exclusão para não haver necessidade de acessar o estado anterior do Banco de Dados.

### 4.1.4 Método Counting

O método Counting [BLA 86, GUP 92] acrescenta a cada relação derivada materializada um novo atributo, o contador, que mantém o número de diferentes derivações para cada tupla da relação derivada. O princípio deste método é incrementar o contador cada vez que uma inserção dá origem a uma nova derivação para a tupla da relação derivada materializada e decrementá-lo quando uma exclusão exclui uma das derivações possíveis. Quando o contador atinge o valor zero, a tupla é excluída da materialização da relação derivada.

Neste método, o *overhead* necessário para manter o contador e a complexidade das solicitações de atualização que devem ser geradas (ver [PAR 94]), são grandes desvantagens em relação ao método DRed. Em vista disso, no Sistema DEDALO, as propagações de atualizações não utilizam o método Counting.

## 4.2 Propagação para relações derivadas definidas por múltiplas regras não recursivas

Um conjunto de regras com a mesma cabeça é interpretado como a união entre os conjuntos de tuplas que podem ser derivados de cada regra. Não é possível determinar sintaticamente se uma relação derivada definida por múltiplas regras satisfaz as propriedades de restrição e segurança. Mesmo se cada uma das regras as satisfazem isoladamente, a união entre elas pode não satisfazê-las. O problema é que cada regra pode associar somente uma derivação para cada tupla da relação derivada, mas a mesma tupla da relação derivada pode ser separadamente derivada por diferentes regras, violando a suposição de uma derivação única [PAR 94].

Os métodos Counting e DRed descritos na seção anterior podem manter eficientemente este tipo de relação derivada. O método DRed cria solicitações de exclusão, inserção e re-inserção para cada regra que define a relação derivada materializada. O exemplo 4.7 demonstra a utilização do método DRed na manutenção de uma relação derivada definida por duas regras.

**Exemplo 4.7:** Considerando a relação derivada materializada *Pacientes_na_UTI*, definida por duas regras não recursivas:

*Pacientes_na_UTI(Nome:X):-pacientes(Nome:X,Setor:'UTI1').*
*Pacientes_na_UTI(Nome:X):-pacientes(Nome:X,Setor:'UTI2').*

Seriam derivadas as seguintes solicitações de atualização pelo método DRed (as solicitações de exclusão e inserção foram derivadas utilizando o método de restrição):

(exclusão) *del Pacientes_na_UTI(Nome:X):-Pacientes_na_UTI(Nome:X), del_pacientes(Nome:X,Setor:'UTI1').*

*del Pacientes_na_UTI(Nome:X) :- Pacientes_na_UTI(Nome:X)*
*del_pacientes(Nome:X,Setor:'UTI2').*

(re-inserção) *ins Pacientes_na_UTI(Nome:X) :-*
*del_Pacientes_na_UTI(Nome:X),*
*Pacientes(Nome:X,Setor:'UTI1').*

*ins Pacientes_na_UTI(Nome:X):-*
*del_Pacientes_na_UTI(Nome:X),*
*Pacientes(Nome:X,Setor:'UTI2').*

(inserção) *ins Pacientes_na_UTI(Nome:X):-*
*ins_pacientes(Nome:X,Setor:'UTI1').*
*ins Pacientes_na_UTI(Nome:X):-*
*ins_pacientes(Nome:X,Setor:'UTI2').*

As solicitações de atualização devem ser executadas na ordem em que foram apresentadas.

## 4.3 Propagação para relações derivadas definidas por regras recursivas

Relações derivadas definidas por regras recursivas podem ser corretamente mantidas com a utilização de três métodos: o método DRed, já citado anteriormente, com algumas modificações, o método PF [HAR 92] e o método LimPro [PAR 94].

### 4.3.1 Método DRed para regras recursivas

O método DRed para regras recursivas deriva as solicitações de exclusão, inserção e re-inserção para cada uma das regras da mesma maneira apresentada nas seções anteriores, utilizando, porém, o método da chave para a criação das solicitações de exclusão. As solicitações devem ser ordenadas de forma que as solicitações de exclusão sejam executadas primeiro, as solicitações de re-inserção a seguir e por último as solicitações de inserção.

As solicitações de exclusão excluem todas as tuplas da relação derivada materializada que poderiam ser derivadas de uma das tuplas removidas durante a transação. Depois de excluir todas as tuplas da relação derivada que poderiam ser derivadas de uma tupla excluída, diretamente ou via uma derivação recursiva, a solicitação de re-inserção é executada. A solicitação verifica para cada tupla da relação derivada que foi removida pela solicitação de exclusão, se existiam derivações alternativas para aquela tupla e, em caso afirmativo, re-insere-a na relação derivada. A solicitação de inserção, finalmente, insere todas as possíveis novas tuplas derivadas das inserções realizadas durante a transação.

O exemplo 4.8 demonstra a utilização do método DRed em relações derivadas definidas por regras recursivas.

**Exemplo 4.8:** Considerando a seguinte relação derivada materializada definida por regras recursivas:

*ancestrais(Ancestral:X,Descendente:Y) :- pacientes(Nome:Y,Pai:X).*

*ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Z) ,*
*ancestrais(Ancestral:Z,Descendente:Y).*

O método DRed criaria as seguintes solicitações de atualização:

(exclusão) *del ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Y),*
*del_pacientes(Nome:Y,Pai:X).*

*del ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Y)*
*del_ancestrais(Ancestral:X,Descendente:Z) ,*
*((ancestrais(Ancestral:Z, Descendente:Y) ,*
*not(ins_ancestrais(Ancestral:Z, Descendente:Y))) ;*
*del_ancestrais(Ancestral:Z, Descendente:Y)).*

*del ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Y),*
*((ancestrais(Ancestral:X, Descendente:Z) ,*
*not(ins_ancestrais(Ancestral:X, Descendente:Z))) ;*
*del_ancestrais(Ancestral:X, Descendente:Z)),*
*del_ancestrais(Ancestral:Z,Descendente:Y) .*

(re-inserção) *ins ancestrais(Ancestral:X,Descendente:Y) :-*
*del_ancestrais(Ancestral:X,Descendente:Y) ,*
*pacientes(Nome:Y,Pai:X).*

*ins ancestrais(Ancestral:X,Descendente:Y) :-*
*del_ancestrais(Ancestral:X,Descendente:Y) ,*
*ancestrais(Ancestral:X,Descendente:Z) ,*
*ancestrais(Ancestral:Z,Descendente:Y).*

(inserção) *ins ancestrais(Ancestral:X,Descendente:Y) :-*
*ins_pacientes(Nome:Y,Pai:X).*

*ins ancestrais(Ancestral:X,Descendente:Y) :-*
*ins_ancestrais(Ancestral:X,Descendente:Z) ,*
*ancestrais(Ancestral:Z,Descendente:Y).*

*ins ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Z) ,*
*ins_ancestrais(Ancestral:Z,Descendente:Y).*

### 4.3.2 Método PF (Propagation-Filtering)

O método PF [HAR 92] é muito similar ao método DRed. A idéia principal é sempre verificar, antes de modificar a materialização, se a modificação é realmente necessária.

Para uma regra:

$p(?_d)$ :- $q_1(?_1)$ ,...,$q_i(?_i)$,...,$q_n(?_n)$, $C$ cf $v$.

onde $p$ é a relação derivada materializada ? é o conjunto de atributos de cada relação, $C$ são as operações de comparação da regra e $v$ é o valor do fator de certeza da regra, o método PF derivaria as seguintes solicitações de atualização para cada literal da regra:

Solicitação de exclusão:

del $p(?_d)$:-$p(?_d)$,$old\_q_1(?_1)$,...,$del\_q_i(?_i)$...,$old\_q_n(?_n)$,$C$,not($aux\_p(?_d)$).

onde $del\_q_i$ é uma relação delta que armazena as tuplas excluídas da relação $q_i$ durante a transação, $old\_q$ representa o conteúdo da relação $q$ no último estado válido antes da transação e $aux\_p$ é um novo literal utilizado para verificar se a tupla deve realmente ser excluída.

Solicitação de inserção:

ins $p(?_d)$ :- $q_1(?_1)$ ,...,$ins\_q_i(?i)$,...,$q_n(?_n)$, $C$, not($p(?_d)$) cf $v$.

onde $ins\_q_i$ é uma relação delta que armazena as inserções feitas sobre a relação $q_i$ durante a transação.

Para verificar se uma inserção está sendo feita corretamente, a solicitação de inserção verifica se a tupla já está na relação derivada materializada. Isto é feito com a inserção do literal not($p(?_d)$) na solicitação de inserção. Isto faz com que apenas as tuplas que ainda não estão na relação $p$ sejam inseridas.

Para verificar se uma exclusão está sendo feita corretamente, é necessário utilizar o literal chamado de $aux\_p$. Este literal representa a relação derivada computada com um mecanismo de avaliação padrão, que necessita acessar as demais relações que derivam a relação $p$. Assim a relação $aux\_p$ seria definida pela seguinte regra:

$aux\_p(?_d)$ :- $q_1(?_1)$ ,...,$q_i(?_i)$,...,$q_n(?_n)$, $C$ cf $v$.

A verificação acrescentada à solicitação de inserção pode ser descartada, pois não acrescenta nada em termos de performance.

O exemplo 4.9 demonstra a utilização do método PF na derivação de solicitações de atualização.

**Exemplo 4.9:** Considerando a seguinte relação derivada materializada definida por regras recursivas:

*ancestrais(Ancestral:X,Descendente:Y) :- pacientes(Nome:Y,Pai:X).*

*ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Z) ,*
*ancestrais(Ancestral:Z,Descendente:Y).*

O método PF derivaria as seguintes solicitações de atualização:

(exclusão) *del ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Y),*
*del_pacientes(Nome:Y,Pai:X),*
*not(aux_ancestrais(Ancestral:X,Descendente:Y)).*

*del ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Y),*
*del_ancestrais(Ancestral:X,Descendente:Z) ,*
*((ancestrais(Ancestral:Z, Descendente:Y) ,*
*not(ins_ancestrais(Ancestral:Z, Descendente:Y))) ;*
*del_ancestrais(Ancestral:Z, Descendente:Y)),*
*not(aux_ancestrais(Ancestral:X,Descendente:Y)).*

*del ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Y),*
*((ancestrais(Ancestral:X, Descendente:Z) ,*
*not(ins_ancestrais(Ancestral:X, Descendente:Z))) ;*
*del_ancestrais(Ancestral:X, Descendente:Z)),*
*del_ancestrais(Ancestral:Z,Descendente:Y),*
*not(aux_ancestrais(Ancestral:X,Descendente:Y)) .*

(inserção) *ins ancestrais(Ancestral:X,Descendente:Y) :-*
*ins_pacientes(Nome:Y,Pai:X),*
*not(ancestrais(Ancestral:X,Descendente:Y)).*

*ins ancestrais(Ancestral:X,Descendente:Y) :-*
*ins_ancestrais(Ancestral:X,Descendente:Z) ,*
*ancestrais(Ancestral:Z,Descendente:Y),*
*not(ancestrais(Ancestral:X,Descendente:Y)).*

*ins ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Z) ,*
*ins_ancestrais(Ancestral:Z,Descendente:Y),*
*not(ancestrais(Ancestral:X,Descendente:Y)).*

A relação *aux_ancestrais* seria definida pelas seguintes regras:

*aux_ancestrais(Ancestral:X,Descendente:Y) :- pacientes(Nome:Y,Pai:X).*

*aux_ancestrais(Ancestral:X,Descendente:Y) :-*
*aux_ancestrais(Ancestral:X,Descendente:Z) ,*
*aux_ancestrais(Ancestral:Z,Descendente:Y).*

A relação *aux_ancestrais* é virtual e só será computada quando a solicitação de exclusão fizer referência a ela. Como neste instante, suas variáveis estarão instanciadas, se o método de derivação utilizado for capaz de derivar apenas as tuplas correspondentes às variáveis instanciadas (derivação top-down) e não toda a relação derivada (derivação bottom-up), a verificação deste método será muito similar à verificação feita no método DRed pela solicitação de re-inserção, com a vantagem de não realizar exclusões desnecessárias. Se, no entanto, o método de derivação utilizado não for capaz de derivar somente as tuplas correspondentes às variáveis instanciadas, o método DRed é mais eficiente.

### 4.3.3 Método LimPro (Limited Prospection)

O método LimPro [PAR 94] é um método que tenta encontrar uma solução intermediária entre os métodos DRed e PF.

Para uma regra:

$p(?_d)$ :- $q_1(?_1)$ ,...,$q_i(?_i)$,...,$q_n(?_n)$, *C cf v.*

onde *p* é a relação derivada materializada ? é o conjunto de atributos de cada relação, *C* são as operações de comparação da regra e *v* é o valor do fator de certeza da regra, o método LimPro derivaria as seguintes solicitações de atualização para cada literal da regra:

Solicitação de exclusão:

*del* $p(?_d)$:-$p(?_d)$,$old\_q_1(?_1)$,...,$del\_q_i(?_i)$...,$old\_q_n(?_n)$,*C, not(fun_p*$(?_d)$*).*

onde $del\_q_i$ é uma relação delta que armazena as tuplas excluídas da relação $q_i$ durante a transação, *old_q* representa o conteúdo da relação *q* no último estado válido antes da transação e *fun_p* é uma função de avaliação que verifica em uma determinada profundidade se a tupla deve ser realmente excluída.

Solicitação de re-inserção:

*ins* $p(?_d)$ :- $del\_p(?_d)$ , $q_1(?)$ ,...,$q_i(?i)$,...,$q_n(?_n)$, *C cf v.*

Solicitação de inserção:

*ins* $p(?_d)$ :- $q_1(?_1)$ ,...,$ins\_q_i(?i)$,...,$q_n(?_n)$, *C cf v.*

onde $ins\_q_i$ é uma relação delta que armazena as inserções feitas sobre a relação $q_i$ durante a transação.

O método LimPro usa uma função de avaliação de relação derivada *fun_p* que acessa as atualizações e as demais relações e detecta se uma tupla pode ser derivada através de uma derivação com profundidade limitada. Esta análise se aplica somente às solicitações de exclusão. A verificação não é exaustiva, mas é capaz de detectar os casos mais relevantes de exclusões incorretas, sem despender recursos em uma investigação de derivações não existentes. Desde que esta pequena análise não é capaz de garantir que uma exclusão é correta, uma fase de re-inserção é necessária.

## 4.4 Propagação para relações derivadas definidas por regras com negação

Em [PAR 94] os literais negados foram classificados em *literais negados limitados* e *literais negados não limitados*. Estes dois conceitos serão descritos a seguir:

? Literais negados limitados: um literal negado é limitado se o conjunto de variáveis do literal negado que aparecem também em um literal positivo contém a chave da relação correspondente.

? Literais negados não limitados: todo literal negado que não é limitado é um literal negado não limitado.

Para regras que possuem literais negados, os mesmos métodos utilizados para regras sem negação podem ser aplicados, apenas algumas modificações devem ser introduzidas para tratar corretamente a negação.

As solicitações de atualização para uma regra contendo um literal negado limitado, seriam obtidas da seguinte forma:

1. Aplica-se um dos métodos apresentados anteriormente

2. Analisa-se as solicitações geradas e se:

2.1. Ao literal negado limitado não for associada uma relação delta (*ins_q* ou *del_q*), mantém-se a solicitação como foi gerada.

2.2. Ao literal negado limitado estiver associada uma relação delta de exclusão, então substitui-se sua ocorrência por uma referência à relação delta de inserção correspondente ao literal negado. Isto é possível, porque a existência de uma inserção em uma relação corresponde a uma exclusão na negação desta mesma relação, ou seja, a partir da inserção de uma determinada tupla em uma relação, a negação desta tupla passa a ser falsa.

2.3. Ao literal negado limitado estiver associada uma relação delta de inserção, então substitui-se sua ocorrência por uma referência à relação delta de exclusão correspondente ao literal negado em conjunção com um literal adicional que é a negação da relação delta de inserção correspondente ao literal. Este literal adicional é

necessário para eliminar as tuplas que foram excluídas e a seguir re-inseridas durante a transação.

O exemplo 4.10 demonstra a derivação das solicitações de atualização para uma regra com um literal negado limitado.

**Exemplo 4.10:** Considerando a seguinte regra que define a relação derivada materializada *Medicamentos_controlados*, onde *Medicamento* é a chave da relação *liberados*:

*Medicamentos_controlados(Medicamento:X) :-*
*medicamentos(Medicamento:X), not(liberados(Medicamento:X)).*

As seguintes solicitações de atualização seriam geradas pela aplicação do método de restrição:

*ins Medicamentos_controlados(Medicamento:X):-*
*ins_medicamentos(Medicamento:X), not(liberados(Medicamentos:X)).*

*ins Medicamentos_controlados(Medicamento:X) :-*
*medicamentos(Medicamento:X),not(ins_liberados(Medicamentos:X)).*

*del Medicamentos_controlados(Medicamento:X):-*
*Medicamentos_controlados(Medicamento:X),*
*del_medicamentos(Medicamento:X).*

*del Medicamentos_controlados(Medicamento:X):-*
*Medicamentos_controlados(Medicamento:X),*
*not(del_liberados(Medicamento:X)).*

Analisando as solicitações geradas, as solicitações definitivas seriam:

*ins Medicamentos_controlados(Medicamento:X):-*
*ins_medicamentos(Medicamento:X),not(liberados(Medicamentos:X)).*

*ins Medicamentos_controlados(Medicamento:X) :-*
*medicamentos(Medicamento:X), del_liberados(Medicamento:X),*
*not(ins_liberados(Medicamento:X)).*

*del Medicamentos_controlados(Medicamento:X):-*
*Medicamentos_controlados(Medicamento:X),*
*del_medicamentos(Medicamento:X).*

*del Medicamentos_controlados(Medicamento:X):-*
*Medicamentos_controlados(Medicamento:X),*
*ins_liberados(Medicamento:X).*

As solicitações de atualização para uma regra contendo um literal negado não limitado, seriam obtidas da seguinte forma:

1. Aplica-se um dos métodos apresentados anteriormente;

2. Analisa-se as solicitações geradas e se:

2.1. Ao literal negado não limitado não for associada uma relação delta (*ins_q* ou *del_q*), mantém-se a solicitação como foi gerada.

2.2. Ao literal negado não limitado estiver associada uma relação delta de exclusão, então substitui-se sua ocorrência por uma referência à relação delta de inserção correspondente ao literal negado.

2.3. Ao literal negado não limitado estiver associada uma relação delta de inserção, então substitui-se sua ocorrência por uma referência à relação delta de exclusão correspondente ao literal negado em conjunção com o literal negado original.

A diferença no tratamento dos literais negados limitados e não limitados está no fato de que para o item 2.3 da análise das solicitações, é necessário verificar em toda a relação correspondente ao literal se realmente não existe uma tupla correspondente à que foi excluída, enquanto para os literais negados limitados, basta conferir se uma tupla correspondente não foi inserida durante a transação. Isto se deve ao fato de que nos literais não limitados, a chave da relação não se encontra entre os atributos que aparecem também nos literais positivos.

O exemplo 4.11 demonstra a derivação das solicitações de atualização para uma regra com um literal negado não limitado.

**Exemplo 4.11:** Considerando a seguinte relação derivada materializada, onde o literal negado *pacientes* é não limitado:

*Homens(Nome:X) :- not(pacientes(Nome:X,Sexo:'F')).*

A aplicação do método DRed geraria as seguintes solicitações de atualização:

*del Homens(Nome:X):- Homens(Nome:X),*
*not(del_pacientes(Nome:X,Sexo:'F')).*

*ins Homens(Nome:X) :- del_Homens(Nome:X),*
*not(pacientes(Nome:X,Sexo:'F')).*

*ins Homens(Nome:X) :- not(ins_pacientes(Nome:X,Sexo:'F')).*

Analisando as solicitações geradas, as solicitações definitivas seriam:

*del Homens(Nome:X):- Homens(Nome:X),ins_pacientes(Nome:X,Sexo:'F').*

*ins Homens(Nome:X) :- del_Homens(Nome:X),*
*not(pacientes(Nome:X,Sexo:'F')).*

*ins Homens(Nome:X) :- del_pacientes(Nome:X,Sexo:'F'),*
*not(pacientes(Nome:X,Sexo:'F')).*

## 4.5 Comentários finais

Existem casos em que uma relação derivada materializada depende de outra relação derivada virtual para ser computada. Como as relações derivadas virtuais estão

sempre atualizadas, pois são computadas quando necessário, se uma atualização causa a inserção ou exclusão de tuplas da relação derivada virtual durante uma transação, as tuplas excluídas e inseridas não aparecerão nas relações delta correspondentes. Como estas informações serão necessárias para a propagação das atualizações e para a detecção de violações de restrições de integridade, como será apresentado no capítulo 6, antes de iniciar a propagação, solicitações de inserção sobre as relações delta correspondentes às relações derivadas virtuais são executadas. A forma destas solicitações de inserção é obtida dos próprios métodos de propagação, como demonstra o exemplo 4.12.

**Exemplo 4.12:** Considerando a seguinte relação derivada virtual, onde o literal *pacientes* é restrito:

*PacientesMaiores(Nome:X) :- pacientes(Nome:X,idade:Y) , Y > 18.*

As seguintes solicitações de inserção sobre as relações delta seriam geradas:

*ins del_PacientesMaiores(Nome:X) :- del_pacientes(Nome:X,Idade:Y) , Y>18.*

*ins ins_PacientesMaiores(Nome:X) :- ins_pacientes(Nome:X,Idade:Y), Y>18.*

A escolha dos métodos adotados pelo sistema DEDALO foi realizada em função da avaliação das seguintes características de cada método:

? Eficiência

? Simplicidade

? Facilidade de implementação

Em função destas características, os métodos de propagação de atualizações escolhidos e implementados no sistema DEDALO, para cada classe de relação derivada, foram os seguintes:

? Para relações derivadas materializadas definidas por uma única regra sem negação:

? Para literais restritos: *método de restrição*

? Para literais seguros: *método da chave*

? Para literais não seguros: *método DRed*, utilizando o método de restrição para a criação das solicitações de exclusão.

? Para relações derivadas materializadas definidas por múltiplas regras não recursivas:

? *Método DRed*, utilizando o método de restrição para a criação das solicitações de exclusão.

? Para relações derivadas materializadas definidas por múltiplas regras recursivas:

? *Método DRed*, utilizando o método da chave para a criação das solicitações de exclusão.

? Para relações derivadas materializadas definidas por regras com negação:

? Análise e modificação das solicitações geradas em função dos literais negados serem *limitados* ou *não limitados.*

No sistema DEDALO, a presença do fator de certeza nas tuplas derivadas, exige um controle adicional ao provido pelos métodos escolhidos. Se uma tupla a ser inserida já se encontra na relação derivada materializada, uma comparação entre os fatores de certeza da tupla já existente e da nova tupla é realizada e se:

? O fator de certeza da nova tupla é menor ou igual ao fator de certeza da tupla já existente, nenhuma ação é realizada, mantendo a tupla antiga na relação derivada;

? O fator de certeza da nova tupla é maior que o fator de certeza da tupla já existente, a nova tupla substitui a antiga.

Os métodos escolhidos mantêm corretamente as relações derivadas materializadas e, na maioria das situações, são muito mais eficientes do que a recomputação da relação derivada. Se a relação derivada, no entanto, é computada com a utilização de funções agregadas, os métodos adotados não funcionam corretamente. Em função disto, se uma relação derivada materializada é definida por pelo menos uma regra que utiliza funções agregadas, toda relação é recomputada a cada modificação nas relações básicas que a derivam, sendo que a relação delta de exclusão conterá todas as tuplas antigas e a relação delta de inserção conterá as novas tuplas.

A decisão de materializar ou não uma relação derivada deve ser cuidadosamente tomada em função dos seguintes fatores:

? Volume de consultas sobre a relação: se a relação é muito consultada, pode ser adequado materializar a relação para diminuir o tempo de acesso;

? Volume de propagações de atualização: se as atualizações sobre as relações básicas afetam com muita freqüência a relação derivada, exigindo grande volume de propagação, pode ser mais adequado manter a relação derivada virtual;

? A existência de funções agregadas nas regras de derivação: se funções agregadas são utilizadas na computação das regras, cada modificação sobre as relações que a derivam provocará a recomputação da relação derivada. Em função disso, se as relações que derivam a relação derivada materializada modificam-se com freqüência, é mais adequado mantê-la virtual.

# 5 Gerenciando atualizações sobre relações derivadas

Atualizações sobre relações derivadas devem ser traduzidas em atualizações sobre as relações básicas, de forma que a atualização solicitada seja visível na relação derivada. Se é solicitada a inserção de uma nova tupla em uma relação derivada, as relações básicas devem ser modificadas de forma que a aplicação das regras de derivação sobre o Banco de Dados modificado, derive a nova tupla cuja inserção foi solicitada. Se é solicitada a exclusão de uma tupla em uma relação derivada, as relações básicas devem ser modificadas de forma que a aplicação das regras de derivação sobre o Banco de Dados modificado, não derive a tupla cuja exclusão foi solicitada. O exemplo 5.1 demonstra esta situação.

**Exemplo 5.1:** Considerando uma relação derivada definida pela seguinte regra:

*pac_resp(Nome:N,Médico:M):-*
*pacientes(Nome:N,Médico_responsável:C) ,*
*médicos(Código:C,Nome:M).*

Supondo que o conteúdo das relações *pacientes* e *médicos* seja o seguinte:

*pacientes('João',1)* *médicos(1,'Pedro')*
*pacientes('Carlos',2)* *médicos(2,'Maria')*
*pacientes('Joana',2)*

e em conseqüência, a relação *pac_resp* tenha as seguintes tuplas:

*pac_resp('João','Pedro')* *pac_resp('Carlos','Maria')*
*pac_resp('Joana','Maria')*

Supondo que fosse solicitada a seguinte inserção:

*ins pac_resp(Nome:'Paulo',Médico:'Maria').*

A única modificação nas relações básicas que tornaria esta tupla visível na relação *pac_resp* seria a inserção da tupla *('Paulo',2)* na relação *pacientes*. Quando apenas uma alternativa de atualização está disponível, a atualização é *determinística*.

Se, no entanto, a seguinte exclusão fosse solicitada:

*del pac_resp(Nome:'João',Médico:'Pedro').*

existiriam três alternativas possíveis: excluir a tupla *('João',1)* da relação *pacientes* ou excluir a tupla *(1,'Pedro')* da relação *médicos* ou excluir ambas as tuplas. Quando mais de uma alternativa está disponível para tornar a solicitação de atualização visível na relação derivada, a atualização é *não determinística.*

O problema das atualizações sobre as relações derivadas em Bancos de Dados Dedutivos está fortemente relacionado com o problema de atualizações sobre visões em sistemas de Bancos de Dados Relacionais [ATZ 92,COS 84,ELM 89,LAN 90,LAU 94, STO 90]. A maioria dos sistemas de BD relacionais comerciais, permitem atualizações somente sobre visões definidas sobre uma única relação. Estas visões são obtidas através das operações de projeção e seleção da álgebra relacional.

Em sistemas de Bancos de Dados Dedutivos, vários estudos sobre o problema das atualizações sobre as relações derivadas [ALV 95, KAK 90, TOM 88, TOM 93, TOR 91] têm buscado romper a limitação imposta pela maioria dos sistemas de BD relacionais, propondo mecanismos que tornem todas as visões atualizáveis.

Na seção 5.1 será apresentada uma discussão sobre as possibilidades de atualização em cada tipo de relação derivada. A seguir, na seção 5.2, um método de geração de traduções das atualizações sobre relações derivadas em atualizações sobre relações básicas, será descrito. Por fim, na seção 5.3, a forma de gerenciamento das atualizações sobre as relações derivadas implementada pelo sistema DEDALO é apresentada.

## 5.1 Análise das possibilidades de atualização das relações derivadas

Em função das operações da álgebra relacional que criam a relação derivada, é possível determinar que relações derivadas são atualizáveis e em que situações. Relações derivadas definidas sobre uma única relação (operações de seleção e projeção) geralmente geram atualizações determinísticas, enquanto relações derivadas definidas sobre mais de uma relação (operações de junção, união, intersecção, produto cartesiano e diferença), geralmente geram atualizações não determinísticas.

A seguir, as possibilidades de atualização para cada operação da álgebra relacional e para regras recursivas serão analisadas.

### 5.1.1 Relações derivadas definidas por uma projeção

Se o literal do corpo da regra é restrito (ver definição na seção 4.1), a relação derivada será sempre atualizável, pois a chave da relação do corpo será sempre conhecida. O exemplo 5.2 demonstra esta situação.

**Exemplo 5.2:** Considerando a seguinte relação derivada definida por uma operação de projeção, onde *Nome* é a chave da relação *pacientes*:

*pacientes_na_UTI(Nome:X) :- pacientes(Nome:X,Setor:'UTI1').*

O literal *pacientes* é restrito. Se a seguinte inserção for solicitada:

*ins pacientes_na_UTI(Nome:'João').*

existirá uma atualização determinística que tornará visível a inserção. Esta atualização será traduzida na seguinte solicitação de inserção:

*ins pacientes(Nome:'João',Setor:'UTI1').*

Se uma exclusão for solicitada, como:

*del pacientes_na_UTI(Nome:'Pedro').*

a atualização também seria determinística, ou seja, seria traduzida em:

*del pacientes(Nome:'Pedro',Setor:'UTI1').*

Se, no entanto, o literal não for restrito, a inserção terá que ter um tratamento adicional, como demonstra o exemplo **5**.3.

**Exemplo 5.3:** Considerando a seguinte relação derivada definida por uma projeção, onde *Nome* é a chave da relação *pacientes*:

*idades(idade:X) :- pacientes(Nome:Y,idade:X).*

onde *pacientes* é um literal não seguro. Se a seguinte inserção fosse solicitada:

*ins idades(idade:12).* ,

a tradução que a tornaria verdadeira seria a inserção de um novo paciente com idade de 12 anos. No entanto, embora os demais atributos possam ser mantidos com valores nulos, a chave da relação deve ser informada, ou seja, é preciso que se obtenha um valor para o atributo *Nome*. Este valor poderia ser fornecido pelo usuário que solicitou a inserção, porém, nem sempre o usuário que tem acesso a uma parte da relação tem conhecimento e permissão para fornecer valores para o restante dela. De fato, uma das principais utilizações de visões é justamente esta: permitir que determinados usuários tenham acesso a apenas uma parte das informações contidas em uma relação, ou seja, esconder informações que não lhe dizem respeito.

Uma outra forma de obter um valor para a chave da relação, seria solicitar ao *projetista da regra*, ou seja, o usuário que definiu a relação derivada e portanto tem acesso a toda relação básica, um valor default para o atributo *Nome* em tempo de

definição da regra. No entanto, apenas um valor default não seria suficiente para tornar todas as projeções atualizáveis. Como no modelo relacional a chave deve ser única, uma segunda solicitação de inserção não poderia utilizar o valor default como conteúdo do atributo *Nome*.

Uma solução melhor seria permitir que o projetista da regra defina, em tempo de definição da relação derivada, um conjunto de regras de atualização que seriam utilizadas na tradução das solicitações de inserção sobre a relação *idades*. Supondo que exista no Banco de Dados a relação *pacientes_não_cadastrados*, que armazena informações sobre pacientes ainda não inseridos na relação *pacientes*. Um exemplo de regras que poderiam ser definidas seria o seguinte:

*ins pacientes(Nome:Y,idade:X):-*
*pacientes_não_cadastrados(Nome:Y,idade:X).*
*del pacientes_não_cadastrados(Nome:Y,idade:X) :-*
*pacientes_não_cadastrados(Nome:Y,idade:X),*
*pacientes(Nome:Y,idade:X).*

que para traduzir a solicitação de inserção *ins idades(idade:12)* seriam utilizadas da seguinte forma:

*ins pacientes(Nome:Y,idade:12):-*
*pacientes_não_cadastrados(Nome:Y,idade:12).*
*del pacientes_não_cadastrados(Nome:Y,idade:12) :-*
*pacientes_não_cadastrados(Nome:Y,idade:12),*
*pacientes(Nome:Y,idade:12).*

onde a relação *pacientes_não_cadastrados* seria a fonte de obtenção de possíveis chaves para a relação *pacientes*. A primeira regra, inseriria em *pacientes* tuplas obtidas da relação *pacientes_não_cadastrados*, cujo valor do atributo *idade* é 12. A segunda regra excluiria da relação *pacientes_não_cadastrados* as tuplas já inseridas na relação *pacientes* pela regra de inserção executada anteriormente.

A definição de regras de atualização, torna qualquer solicitação de inserção sobre uma relação derivada definida por um literal não restrito realizável. Com a definição de regras de atualização deste tipo, a inserção sobre a relação derivada *idades* somente não seria possível se não houvessem tuplas disponíveis na relação *pacientes_não_cadastrados*. No entanto, a responsabilidade de que as regras definidas como tradução para a atualização traduzem corretamente a inserção solicitada, é do projetista da regra.

Por outro lado, uma exclusão sobre a relação *idades*, como:

*del idades(idade:15).*

seria determinística e poderia ser traduzida por:

*del pacientes(idade:15).*

Assim, inserções sobre relações derivadas definidas por uma operação de projeção poderão ser automaticamente traduzidas se o literal do corpo for restrito. Se o literal não for restrito, o valor da chave durante uma inserção deverá ser fornecido. Obviamente, se a tradução dá origem à inserção de uma tupla que viola restrições de integridade, como chave duplicada, por exemplo, a solicitação de inserção não será realizada. As exclusões sempre poderão ser automaticamente traduzidas.

### 5.1.2 Relações derivadas definidas por uma seleção

Solicitações de atualização sobre relações definidas por uma operação de seleção poderão ser sempre automaticamente traduzidas. O exemplo **5**.4 demonstra as atualizações sobre uma relação derivada definida por uma operação de seleção.

**Exemplo 5.4:** Considerando a relação derivada *pacientes_maiores*, definida por uma operação de seleção:

*pacientes_maiores(Nome:X,Setor:Y,idade:Z) :-*
*pacientes(Nome:X,Setor:Y,idade,Z) , Z >= 18.*

A solicitação de inserção:

*ins pacientes_maiores(Nome:'Maria',Setor:'Maternidade',idade:25).*

seria traduzida como:

*ins pacientes(Nome:'Maria',Setor:'Maternidade',idade:25).*

Uma solicitação de inserção somente não seria realizada se a tupla a ser inserida viola alguma restrição de integridade ou se não satisfaz a condição de seleção, como:

*ins pacientes_maiores(Nome:'Maria',Setor:'Maternidade',idade:12).*

Não existe tradução que torne a tupla visível na relação *pacientes_maiores*. As exclusões, no entanto, sempre serão traduzíveis. Por exemplo, a solicitação:

*del pacientes_maiores(Nome:'João',Setor:'UTI1',idade:19).*

seria traduzida como:

*del pacientes(Nome:'João',Setor:'UTI1',idade:19).*

### 5.1.3 Relações derivadas definidas por uma junção entre tabelas

Atualizações sobre relações derivadas definidas por uma operação de junção geralmente são não determinísticas e por isso são proibidas pela maioria dos Sistemas de Banco de Dados Relacionais. O exemplo **5**.5 demonstra o que acontece com as atualizações sobre relações derivadas definidas por uma operação de junção entre tabelas.

**Exemplo 5.5:** Considerando a seguinte relação derivada, definida por uma operação de junção:

*pac_resp(Nome:N,CodMed:C,Médico:M):-*
*pacientes(Nome:N,Médico_responsável:C) ,*
*médicos(Código:C,Nome:M).*

Se fosse solicitada a seguinte inserção:

*ins pac_resp(Nome:'João',CodMed:4,Médico:'Antônio').*

dependendo do conteúdo do Banco de Dados, as traduções possíveis seriam:

*ins pacientes(Nome:'João',Médico_responsável:4).* , se já existe uma tupla *médicos(4 ,'Antônio')*;

*ins médicos(Código:4,Nome:'Antônio').* , se já existe uma tupla *pacientes('João',4)* ou

*ins pacientes(Nome:'João',Médico_responsável:4).* e *ins médicos(4,'Antônio').* se nenhuma das tuplas existem no BD atual.

Se a seguinte exclusão fosse solicitada:

*del pac_resp(Nome:'João',CodMed:1, Médico:'Pedro').*

existiriam três traduções possíveis:

*del pacientes(Nome:'João',Médico_responsável:1).*

*del médicos(Código:1,Nome:'Pedro').* ou

*del pacientes(Nome:'João',Médico_responsável:1).* e
*del médicos (Código:1,Nome:'Pedro').*

Muitos sistemas de Bancos de Dados Dedutivos eliminariam a última alternativa, adotando uma política de *atualizações mínimas*, ou seja, escolhem a alternativa que provoca o menor número de modificações na base de dados. Adotando esta política, o problema se resume a escolher uma das relações do corpo para realizar a exclusão.

Um critério que pode ser utilizado nesta escolha é a *integridade referencial*, ou seja, se a chave de uma relação é chave estrangeira em alguma outra relação do Banco de Dados, procura-se não realizar a exclusão nesta relação para que não fiquem tuplas órfãs ou seja desencadeada uma exclusão em cascata das tuplas órfãs. No

entanto, ainda podem haver casos onde ambas as chaves das relações envolvidas na junção são chave estrangeira em outras relações do banco. Nestes casos, como acontecia na operação de projeção, o projetista da regra deve informar, em tempo de definição da relação derivada, quais as regras que traduzem corretamente as atualizações sobre a relação derivada. Para a relação derivada do exemplo **5**.5, uma possível tradução seria:

*del pacientes(Nome:N,Médico_responsável:M).* , ou seja, em caso de exclusão sobre a relação *pac_resp*, excluir a tupla correspondente na relação *pacientes*.

### 5.1.4 Relações derivadas definidas por um produto cartesiano

A operação de produto cartesiano tem comportamento idêntico ao da operação de junção, como demonstra o exemplo 5.6.

**Exemplo 5.6:** Considerando a seguinte relação derivada, definida por uma operação de produto cartesiano:

*setores_medicamentos(setor:X,medicamento:Y) :- setores(setor:X) ,*
*medicamentos(medicamento:Y).*

Supondo que as seguintes operações de inserção e exclusão fossem solicitadas:

*ins setores_medicamentos(setor:'UTI1',medicamento:'aspirina').*
*del setores_medicamentos(setor:'UTI1',medicamento:'digoxina').*

Exatamente como na operação de junção, a inserção seria traduzida em uma inserção sobre a relação *setores* ou sobre a relação *medicamentos* se em uma delas não houvesse uma tupla correspondente, ou em ambas as relações se não houvessem as tuplas necessárias em nenhuma delas.

Da mesma forma, a exclusão seria traduzida em uma exclusão sobre a relação *setores* ou em uma exclusão sobre a relação *medicamentos*, de acordo com as restrições de integridade referencial existentes e com as possíveis regras de tradução informadas pelo projetista da regra

### 5.1.5 Relações derivadas definidas por uma diferença

As inserções e exclusões sobre relações derivadas definidas por uma operação de diferença são tratadas da mesma forma que nas operações de junção e produto cartesiano, com uma pequena modificação em função da presença da negação. O exemplo **5**.7 demonstra esta situação.

**Exemplo 5.7:** Considerando a seguinte relação derivada definida por uma operação de diferença:

*Medicamentos_controlados(Medicamento:X) :-*
*medicamentos(Medicamento:X), not(liberados(Medicamento:X)).*

Se fossem solicitadas as seguintes atualizações:

*ins Medicamentos_controlados(Medicamento:'digoxina').*
*del Medicamentos_controlados(Medicamento:'antidepressivo').*

Como nas operações de junção e produto cartesiano, a inserção seria traduzida em atualizações sobre a relação *medicamentos*, sobre a relação *liberados*, ou sobre ambas. A diferença é que, caso a relação *liberados* não satisfaça a regra, a inserção deve ser substituída por uma exclusão. Como o literal *liberados* é negativo, se ele não satisfaz a regra, deve existir uma tupla *liberados('digoxina')*, e, para que a regra seja satisfeita, esta tupla deve ser excluída. Assim, as possíveis traduções para a inserção na relação *Medicamentos_controlados* seriam:

*ins medicamentos(Medicamento:'digoxina').*

*del liberados(Medicamento:'digoxina').* ou

*ins medicamentos(Medicamento:'digoxina').* e
*del liberados(Medicamento:'digoxina').*

No caso da exclusão, o princípio é o mesmo: substitui-se uma exclusão na relação *liberados* por uma inserção. Assim, as possíveis traduções para a exclusão na relação *Medicamentos_controlados* seriam:

*del medicamentos(Medicamento:'antidepressivo').*

*ins liberados(Medicamento:'antidepressivo').* ou

*del medicamentos(Medicamento:'antidepressivo').* e
*ins liberados(Medicamento:'antidepressivo').*

Da mesma forma que nas operações de junção e produto cartesiano, o projetista da regra pode definir uma tradução caso não exista uma alternativa satisfatória.

### 5.1.6 Relações derivadas definidas por uma união

Se uma relação derivada é definida por uma operação de união, uma inserção sobre esta relação deverá ser traduzida em uma inserção sobre uma das relações que participam da união. Caso uma exclusão seja solicitada, esta deve ser traduzida em exclusões de tuplas das relações participante da união. O exemplo 5.8 demonstra esta situação.

**Exemplo 5.8:** Considerando a seguinte relação derivada definida por operação de união:

*funcionários(Código:X,Nome:Y) :- médicos(Código:X,Nome:Y) ;*
*enfermeiros(Código:X,Nome:Y).*

Se fosse solicitada a seguinte inserção:

*ins funcionários(Código:10,Nome:'Ana').*

As traduções possíveis seriam:

*ins médicos(Código:10,Nome:'Ana').*

*ins enfermeiros(Código:10,Nome:'Ana').* ou

*ins médicos(Código:10,Nome:'Ana').* e
*ins enfermeiros(Código:10,Nome:'Ana').*

Esta atualização é não determinística, pois existe mais de uma alternativa. Qualquer uma irá derivar a nova tupla e portanto é uma tradução correta. No entanto, apenas uma refletirá a verdade, ou seja, se a funcionária Ana é médica ou enfermeira. No sistema DEDALO, a presença de fatores de certeza pode ser utilizada para determinar que inserção será realizada, ou seja, a que produzir a tupla com o maior fator. Mesmo assim, poderia ocorrer um empate entre os fatores e a atualização continuaria não determinística. Este indeterminismo da operação de união na inserção, exige que o projetista da regra defina uma tradução para as inserções na relação derivada.

Caso a seguinte exclusão fosse solicitada:

*del funcionários(Código:15,Nome:'Marcos').*

A tradução seria determinística, pois bastaria excluir da relação *médicos* se a tupla correspondente estivesse naquela relação, ou da relação *enfermeiros,* ou ainda de ambas, se existissem tuplas correspondentes nas duas relações.

### 5.1.7 Relações derivadas definidas por uma intersecção

As solicitações de inserção e atualização em relações derivadas definidas por operações de intersecção teriam tratamento semelhante às atualizações sobre relações definidas por operações de junção e produto cartesiano: insere tuplas em uma ou em ambas as relações, conforme necessário, e exclui de uma das relações de acordo com as restrições de integridade referencial ou uma possível tradução fornecida pelo projetista da regra. O exemplo 5.9 demonstra as atualizações sobre as relações derivadas definidas por uma operação de intersecção.

**Exemplo 5.9:** Considerando a seguinte relação derivada, que armazena os funcionários que são médicos e enfermeiros ao mesmo tempo:

*médicos_enfermeiros(Código:X,Nome:Y) :- médicos(Código:X,Nome:Y) ,*
*enfermeiros(Código:X,Nome:Y).*

Se as seguintes operações de atualização fossem solicitadas:

*ins médicos_enfermeiros(Código:16,Nome:'Marta').*

*del médicos_enfermeiros(Código:17,Nome:'Júlio').*

As traduções possíveis seriam:

*ins médicos(Código:16,Nome:'Marta').*

*ins enfermeiros(Código:16,Nome:'Marta').* ou

*ins médicos(Código:16,Nome:'Marta').* e
*ins enfermeiros(Código:16,Nome:'Marta').*

e para a exclusão:

*del médicos(Código:17,Nome:'Júlio').*

*del enfermeiros(Código:17,Nome:'Júlio').* ou

*del médicos(Código:17,Nome:'Júlio').* e
*del enfermeiros(Código:17,Nome:'Júlio').*

### 5.1.8 Relações derivadas recursivas

Relações derivadas recursivas sempre são definidas por operações de união, pois é necessário pelo menos duas regras para criar uma relação recursiva não vazia e pelo menos uma destas regras não será recursiva. A estratégia adotada em relações recursivas é realizar as solicitações de inserção nas regras não recursivas. O exemplo 5.10 demonstra esta situação.

**Exemplo 5.10:** Considerando a seguinte relação derivada definida por regras recursivas:

*ancestrais(Ancestral:X,Descendente:Y) :- pacientes(Nome:Y,Pai:X).*

*ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Z) ,*
*ancestrais(Ancestral:Z,Descendente:Y).*

Se fosse solicitada a seguinte inserção:

*ins ancestrais(Ancestral:'Adão',Descendente:'Abel').*

A tradução seria feita sobre a regra não recursiva, ou seja:

*ins pacientes(Nome:'Abel',Pai:'Adão').*

Esta tradução tornará visível a tupla na relação *ancestrais*, e portanto é uma tradução correta. No entanto, não se pode ter certeza de que a tradução escolhida reflete a verdade e, portanto, o projetista da regra pode informar uma outra tradução através da definição de um conjunto de regras de atualização.

Se for solicitada a seguinte exclusão:

*del ancestrais(Ancestral:'Pedro',Descendente:'Paulo').*

É necessário encontrar todas as tuplas da relação *pacientes* que derivam a tupla que se deseja excluir e então excluí-las. Se existir uma tupla *pacientes('Pedro','Paulo')*, a exclusão pode ser feita diretamente, se não, é necessário substituir as constantes na regra recursiva, obtendo uma nova solicitação de exclusão, ou seja:

*del ancestrais(Ancestral:'Pedro',Descendente:Z):-*
*ancestrais(Ancestral:'Pedro',Descendente:Z),*
*ancestrais(Ancestral:Z,Descendente:'Paulo').*

Esta solicitação de exclusão geraria um novo conjunto de tuplas que devem ser excluídas, ou seja, todas as tuplas da relação *ancestrais* que satisfazem o corpo da regra de exclusão. A exclusão destas tuplas, por sua vez, deveria ser traduzida em novas solicitações de exclusão, até que sejam encontradas as tuplas da relação *pacientes* que, uma vez excluídas, tornarão a derivação da tupla *ancestrais('Pedro','Paulo')* impossível.

A escolha da alternativa de exclusão ou a criação de uma nova tradução é responsabilidade do projetista da regra.

## 5.2 Geração de traduções para atualizações sobre relações derivadas

O gerenciamento das atualizações sobre as relações derivadas no sistema DEDALO é feito da seguinte forma: para cada relação derivada o sistema, mais especificamente, o módulo gerenciador de regras, descrito no capítulo 8, deriva um conjunto de traduções possíveis que são apresentadas ao projetista da regra. O projetista pode escolher uma das traduções geradas, caso exista não determinismo, ou criar sua própria tradução através de um conjunto de regras de atualização. Nesta seção, será descrito o procedimento utilizado pelo gerenciador de regras para gerar as traduções possíveis.

### 5.2.1 Geração de traduções para inserções

O processo de geração de traduções para inserções irá gerar para cada regra que define uma relação derivada, uma regra de inserção (ou de exclusão, no caso dos literais negativos) para cada literal.

Na tradução de uma inserção, podem ocorrer duas situações: apenas alguns dos literais não possuem tuplas que satisfazem a regra ou nenhum dos literais a satisfaz. Supondo que exista uma condição de igualdade entre um atributo do literal $l_1$ e do literal $l_2$, para que a regra seja satisfeita, estes dois literais devem ter o mesmo valor para aquele atributo. Se o atributo estiver presente também na cabeça da regra, não haverá problema, pois o valor do atributo será fornecido na solicitação de inserção sobre a relação derivada. Se o literal $l_1$ possuir tuplas que satisfazem a regra, o valor para o atributo em condição de igualdade de $l_2$ pode ser obtido das tuplas de $l_1$ e vice-versa. Caso nenhum dos literais possua tuplas que satisfaçam a regra, a inserção não poderá ser traduzida. O exemplo 5.11 demonstra esta situação.

**Exemplo 5.11:** Considerando a seguinte relação derivada:

*MedResp(Médico:M,Paciente:P) :- pacientes(Nome:P,CodMed:C),*
*médicos(Código:C,Nome:M).*

Se a fosse solicitada a seguinte inserção:

*ins MedResp(Médico:'Pedro',Paciente:'João').*

Se já existisse uma tupla *pacientes('João',C)* na relação *pacientes*, o valor do atributo *Código* da relação *médicos* seria obtido da tupla de *pacientes* e a inserção na relação *médicos* seria possível. O mesmo ocorreria caso houvesse uma tupla que satisfizesse a regra em *médicos*. Se nenhuma das relações possuírem tuplas que satisfaçam a regra, no entanto, a inserção não será possível, pois não seriam conhecidos valores para os atributos *CodMed* e *Código*.

Em função desta situação, somente se saberá se existe uma tradução para muitas solicitações de inserção em run-time. O mesmo ocorre quando atributos que não estão na cabeça da regra, nem em condição de igualdade com constantes, estão envolvidos em operações de comparação. É fundamental obter um valor para estes atributos, mas a menos que exista uma tupla que satisfaça a regra e que contenha este atributo, a inserção não poderá ser realizada.

Assim, a tradução de uma inserção irá gerar solicitações de inserção para os literais restritos cujos atributos envolvidos em operações de comparação estejam também no conjunto de atributos restritos, pois estes não dependem de outros literais para serem inseridos. A geração de solicitações de inserção para os demais literais irá depender do estado do Banco de Dados no momento da atualização, quando será verificado se um conjunto de solicitações de inserção que satisfaça a regra pode ser gerado a partir das tuplas existentes no Banco de Dados e das tuplas que podem ser inseridas nos literais restritos. Se tal conjunto não puder ser encontrado, a inserção não será realizada e as inserções sobre os literais restritos não serão feitas.

O algoritmo que gera as traduções é o seguinte:

**Algoritmo 3:**

Considerando uma regra de derivação genérica não recursiva sem funções agregadas:

$p(?_d) :- q_1(?_1) ,...,q_i(?_i),...,q_n(?_n), C.$

onde *p* é a relação derivada, ? é o conjunto de atributos de cada relação e *C* são as operações de comparação da regra.

Para uma solicitação de inserção:

*ins p(? ).*

onde ? é um conjunto de constantes, seria gerada uma solicitação de atualização para cada literal do corpo, da seguinte forma:

1. Se o literal é positivo:

1.1. Se todos os atributos do literal cujo valor não pode ser nulo estão no conjunto de atributos restritos e se nenhum atributo do literal estiver envolvido em

operações de comparação ou em condições de igualdade com outro atributo, ou se todos os atributos envolvidos em operações de comparação ou em condições de igualdade com outro atributo, estão no conjunto de atributos restritos, uma solicitação de inserção

da seguinte forma é gerada:

$$ins\ q_i(?_i).$$

onde $q_i$ é o literal para o qual se está gerando a solicitação de inserção e $?_i$ são as constantes referentes aos atributos do literal $q_i$ que aparecem na cabeça da regra e as constantes com as quais atributos de $q_i$ estão envolvidos em condições de igualdade.

1.2. Se algum atributo do literal não puder permanecer com valor nulo por fazer parte da chave ou por estar envolvido em uma operação de comparação ou em uma condição de igualdade e este atributo não estiver no conjunto de atributos restritos, mas puder ser obtido através de outras tuplas, então a solicitação de inserção terá a seguinte forma:

$$ins\ q_i(?_i) :- ins\_p(?), q_j(?_j), q_l(?_l), C_i.$$

onde $?_i$ são os atributos de $q_i$ cujos valores podem ser obtidos das tuplas das relações $q_j$ e $q_l$, $ins\_p(?)$ é a tupla que se deseja inserir e $C_i$ são as operações de comparação entre atributos de $q_i$ e constantes.

1.3. Se não for possível gerar uma solicitação de inserção nos passos 1.1 e 1.2, então as inserções sobre o literal não poderão ser feitas. Nestes casos, a tradução da solicitação de inserção só terá sucesso, se não for necessário inserir neste literal para que a regra seja satisfeita.

2. Se o literal é negativo:

2.1. Se o literal é restrito, a seguinte solicitação de exclusão seria gerada:

$$del\ \ q_i(?_i) :- q_i(?_i), C_i.$$

onde $q_i$ é o literal negativo, $?_i$ são as constantes referentes aos atributos do literal $q_i$ que aparecem na cabeça da regra e as constantes com as quais atributos de $q_i$ estão envolvidos em condições de igualdade e $C_i$ são as operações de comparação entre constantes e as variáveis do literal.

2.2. Se o literal não é restrito, a seguinte solicitação de exclusão seria gerada:

$$del\ \ q_i(?_1) :- q_i(?_i), q_1(?_1),...,q_n(?_n), C_i.$$

onde $q_i$ é o literal negativo, $q_1...q_n$ são os literais do corpo da regra que possuem variáveis ligadas a variáveis de $q_i$ e $C_i$ são as operações de comparação em que as variáveis dos literais que aparecem na regra de exclusão gerada estão envolvidas.

Os fatores de certeza das tuplas inseridas será 1. A determinação de quais solicitações de inserção geradas serão executadas dependerá do estado do Banco de Dados no momento da atualização. Depois que todas as possíveis solicitações de inserção e exclusão forem executadas, o sistema verificará se as atualizações foram suficientes para derivar a tupla na relação derivada. Se a tupla não for derivada, as atualizações realizadas são desfeitas e a solicitação de inserção sobre a relação derivada não pode ser atendida.

O passo 1.1 do algoritmo faz uso das condições de igualdade entre os atributos para obter valores para atributos que não estão na cabeça da regra, como demonstra o exemplo 5.12.

**Exemplo 5.12:**

*PacientesNaUTI(Nome:X) :- pacientes(Nome:X,Setor:Y) , Y='UTI'.*

A seguinte solicitação de inserção seria gerada para o literal *pacientes*:

*ins pacientes(Nome:X,Setor:'UTI').*

O passo 1.2 do algoritmo se refere a situações onde o valor de um atributo, apesar de não fazer parte da chave da relação, é essencial para que a regra seja satisfeita e, conseqüentemente, a tupla que se deseja inserir possa ser derivada.

O passo 2.1 utiliza as operações de comparação entre constantes e as variáveis do literal para limitar as tuplas que devem ser excluídas, como demonstra o exemplo 5.13.

**Exemplo 5.13:**

*MedicamentosControlados(Medicamento:X) :-*
*medicamentos(Medicamento:X),*
*not(liberados(Medicamento:X)) , X <> 'digoxina'.*

Para o literal *liberados* a seguinte solicitação de exclusão seria gerada:

*del liberados(Medicamento:X) :- liberados(Medicamento:X),X <> 'digoxina'.*

Esta solicitação de exclusão não excluiria as tuplas que não satisfizessem a operação de comparação. Como as tuplas que não satisfizerem a operação de comparação não serão utilizadas na derivação da relação *MedicamentosControlados*, não há necessidade de excluí-las.

O passo 2.2 utiliza outros literais do corpo da regra para gerar uma solicitação de exclusão sobre o literal negado não restrito, porque a chave deste literal não é conhecida e portanto, é necessário que os demais literais sejam utilizados para instanciar as variáveis do literal negado e assim excluir as tuplas necessárias, como demonstra o exemplo 5.14.

**Exemplo 5.14:**

*MédicosSemPacientes(Nome:X) :- médicos(Código:Y,Nome:X) ,*
*not(pacientes(Médico_responsável:Y)).*

Para o literal *pacientes* a seguinte solicitação de exclusão seria gerada:

*del pacientes(Médico_responsável:Y):- pacientes(Médico_responsável:Y),*
*médicos(Código:Y,Nome:X).*

Todas as tuplas da relação *pacientes* que satisfazem a regra, serão excluídas.

### 5.2.2 Geração de traduções para exclusões

O processo de geração de traduções para solicitações de exclusão sobre relações derivadas irá gerar solicitações de exclusão (inserção, para os literais negativos) para cada literal de cada regra que deriva a relação. O algoritmo que gera as traduções é o seguinte:

**Algoritmo 4:**

Considerando uma regra de derivação genérica:

$p(?_d)$ :- $q_1(?_1)$ ,...,$q_i(?_i)$,...,$q_n(?_n)$, $C$.

onde $p$ é a relação derivada, $?_i$ é o conjunto de atributos de cada relação e $C$ são as operações de comparação da regra. Para uma solicitação de exclusão:

*del* $p(?)$.

onde $?$ é um conjunto de constantes, seria gerada uma solicitação de atualização para cada literal do corpo, da seguinte forma:

1. Se a cabeça da regra não possui funções agregadas:

1.1. Se o literal é positivo:

1.1.1. Se o literal é restrito:

1.1.1.1. Se não existem operações aritméticas na cabeça da regra ou as variáveis do literal não estão envolvidas nestas operações aritméticas, a seguinte solicitação de exclusão seria gerada:

*del* $q_i(?_i)$ :- $q_i(?_i)$,$C_i$.

onde $q_i$ é o literal, $?_i$ são as constantes referentes aos atributos do literal $q_i$ que aparecem na cabeça da regra, as constantes com as quais atributos de $q_i$ estão envolvidos em condições de igualdade e as variáveis de $q_i$ que estão envolvidas nas operações de comparação $C_i$.

1.1.1.2. Se existem variáveis do literal envolvidas em operações aritméticas na cabeça da regra, a seguinte solicitação de exclusão seria gerada:

*del* $q_i(?_i)$:- $q_i(?_i)$,$q_1(?_1)$ ,...,$q_n(?_n)$, $C$, $Const = Exp$.

onde $q_i$ é o literal que está sendo analisado, $q_1 ... q_n$ são os demais literais do corpo da regra, $C$ são as operações de comparação da regra, *Const* é a constante informada como o resultado da operação aritmética na solicitação de exclusão e *Exp* é a operação aritmética da cabeça da regra.

1.1.2. Se o literal não é restrito:

1.1.2.1. Se não existem operações aritméticas na cabeça da regra ou as variáveis do literal não estão envolvidas nestas operações aritméticas, a seguinte solicitação de exclusão seria gerada:

$$del\ \ q_i(?_i) :- q_i(?_i), q_1(?_1), ..., q_n(?_n), C_i.$$

onde $q_i$ é o literal que está sendo avaliado, $q_1 ... q_n$ são os literais do corpo da regra que possuem variáveis ligadas a variáveis de $q_i$ e $C_i$ são as operações de comparação em que as variáveis dos literais que aparecem na regra de exclusão gerada estão envolvidas.

1.1.2.2. Se existem variáveis do literal envolvidas em operações aritméticas na cabeça da regra, a seguinte solicitação de exclusão seria gerada:

$$del\ \ q_i(?_i) :- q_i(?_i), q_1(?_1), ..., q_n(?_n), C, Const = Exp.$$

onde $q_i$ é o literal que está sendo analisado, $q_1 ... q_n$ são os demais literais do corpo da regra, *C* são as operações de comparação da regra, *Const* é a constante informada como o resultado da operação aritmética na solicitação de exclusão e *Exp* é a operação aritmética da cabeça da regra.

1.2. Se o literal é negativo, uma solicitação de inserção será gerada de acordo com o passo 1.1 do algoritmo 3.

2. Se a cabeça da regra possui funções agregadas:

2.1. Se o literal possui variáveis envolvidas na função agregada, a seguinte solicitação de exclusão será gerada:

$$del\ q_i(?_i) :- q_i(?_i), q_1(?_1), ..., q_n(?_n), C.$$

onde $q_i$ é o literal que está sendo analisado, $q_1 ... q_n$ são os demais literais do corpo da regra e *C* são as operação de comparação do corpo da regra.

2.2. Se o literal não possui variáveis envolvidas na função agregada, a solicitação de exclusão não será gerada.

O passo 1.1.1.2 utiliza os outros literais do corpo para que todas as variáveis envolvidas na operação aritmética possam estar instanciadas no momento de sua avaliação, como demonstra o exemplo **5**.15.

**Exemplo 5.15:**

*TotalConta(Nome:X,Valor:Y*Z) :- Quartos(Tipo:Q,Diária:Y) ,*

*internação(Pacientes:X,Quarto:Q,Dias:Z).*

A seguintes solicitações de exclusão seriam geradas:

*del Quartos(Tipo:Q,Diária:Y) :- Quartos(Tipo:Q,Diária:Y)*
*internação(Pacientes:X,Quarto:Q,Dias:Z), Valor = Y*Z.*

*del internação(Pacientes:X,Quarto:Q,Dias:Z) :-*
*internação(Pacientes:X,Quarto:Q,Dias:Z),*
*Quartos(Tipo:Q,Diária:Y), Valor = Y*Z.*

A variável *Valor*, no momento da execução da solicitação de exclusão estará instanciada com a constante associada ao atributo *Valor* da relação derivada *TotalConta*. Se a solicitação de exclusão considerasse apenas uma das relações (*Quartos* ou *Internações*), uma das variáveis (*Y* ou *Z*) não estariam instanciadas no momento da avaliação da expressão. Assim, todas as tuplas em que a aplicação da operação aritmética sobre os valores de seus atributos resulta no valor informado na solicitação de exclusão sobre a relação *TotalConta*, serão excluídas.

O passo 1.1.2.1 utiliza os demais literais do corpo da regra para limitar o número de tuplas que devem ser excluídas, pois o literal para o qual se está gerando a solicitação de exclusão não é restrito, e, portanto, sua chave não é conhecida. O exemplo **5**.16 demonstra esta situação.

**Exemplo 5.16:**

*Pacientes_na_UTI(Nome:X) :- pacientes(Nome:X,Setor:Y) ,*
*Setores(Código:Y,Descrição:'UTI').*

A seguinte solicitação de exclusão seria gerada para o literal não restrito *Setores*:

*del Setores(Código:Y,Descrição:'UTI'):-*
*Setores(Código:Y,Descrição:'UTI'),*
*pacientes(Nome:X,Setor:Y).*

Se não fossem utilizados os demais literais do corpo, todos os setores cuja *Descrição* é *'UTI'* seriam excluídos. Desta forma, somente os setores que têm uma tupla correspondente na relação *pacientes* e cuja *Descrição* é *'UTI'* seriam excluídos.

O passo 2.1 excluiria todas as tuplas que estão envolvidas na computação da função agregada, como demonstra o exemplo 5.17.

**Exemplo 5.17:**

*PacientesPorSetor(Setor:X,Total:count(Y)) :- pacientes(Nome:Y,Setor:Z),*
*setores(Código:Z,Descrição:X).*

A seguinte solicitação de exclusão seria gerada para o literal *pacientes*:

*del pacientes(Nome:Y,Setor:Z):-pacientes(Nome:Y,Setor:Z),*
*setores(Código:Z,Descrição:X).*

## 5.3 Gerenciamento das atualizações sobre as relações derivadas no sistema DEDALO

O módulo *Gerenciador de Regras* do sistema DEDALO é uma ferramenta para criação de regras que dão origem a relações derivadas. Neste módulo, através de uma interação com o projetista da regra, as traduções de atualizações sobre relações derivadas são definidas.

No sistema DEDALO, o projetista da regra pode escolher, no momento da definição da relação derivada, se ela poderá ser atualizada ou não. Se a relação for definida como atualizável, um conjunto de solicitações de inserção e um conjunto de solicitações de exclusão serão gerados para cada regra que define a relação derivada. O projetista da regra, então, poderá optar entre utilizar as traduções geradas ou criar um novo conjunto de regras de inserção e exclusão, que tornarão a atualização sobre a relação derivada visível. A responsabilidade sobre a correção desta nova tradução é do projetista da regra. Nas situações em que o sistema não foi capaz de gerar nenhuma tradução, o projetista deve criar a sua tradução.

Caso o projetista prefira utilizar as traduções geradas pelo sistema, deverá tomar algumas decisões:

1. Em relações derivadas definidas por mais de uma regra, o projetista deverá escolher a tradução para inserção gerada para uma delas. Isto é necessário porque em uma operação de união, somente uma regra necessita ser satisfeita para que a tupla derivada seja visível. No momento da execução, todas as traduções serão aplicadas sobre o banco e a que gerar o maior fator de certeza será executada. Caso exista um empate, a tradução escolhida pelo projetista será executada.

2. Em relações derivadas que são formadas por mais de uma regra, o projetista deverá escolher uma tradução de exclusão gerada, para cada regra que forma a relação derivada. Isto é necessário porque excluindo tuplas de apenas um dos literais do corpo, a tupla que se deseja excluir não será mais derivada. Para auxiliar nesta escolha, informações sobre a existência de restrições de integridade referencial são fornecidas.

As alterações sobre tuplas são consideradas como exclusões seguidas de inserções das tuplas alteradas.

O processamento das solicitações de atualização sobre as relações derivadas é o seguinte:

1. Quando uma solicitação de inserção ou exclusão sobre uma relação derivada é submetida, o conjunto de todas as tuplas que devem ser inseridas ou excluídas da relação derivada é obtido.

2. Para cada tupla a ser inserida, verifica-se se a mesma já se encontra na relação derivada. Em caso afirmativo, a tupla não é inserida.

3. Para cada tupla a ser excluída, verifica-se se a mesma realmente se encontra na relação derivada. Se a tupla não for encontrada, a exclusão não é realizada.

4. Se a atualização solicitada é uma inserção, é adotado o seguinte procedimento:

4.1. Para cada solicitação de inserção gerada, as variáveis da cabeça da regra de inserção são instanciadas seguindo o seguinte algoritmo:

4.1.1. As variáveis da cabeça da regra de inserção que estão ligadas à cabeça da regra de derivação, são instanciadas com as constantes da tupla a ser inserida na relação derivada.

4.1.2. As variáveis do corpo da regra de inserção em condições de igualdade com constantes ou com variáveis já instanciadas, recebem os valores associados.

4.1.3. Repete-se o passo 4.1.2 até que nenhuma nova variável possa ser instanciada.

4.2. Para cada solicitação de exclusão gerada em função da presença de literais negativos, são executados os passos 5.1.1 a 5.1.3.

4.3. Cada solicitação de inserção que gera novas tuplas é executada.

4.4. Cada solicitação de exclusão referente aos literais negativos que exclui tuplas, é executada.

5. Se a solicitação de atualização é uma exclusão é adotado o seguinte procedimento:

5.1. Para cada solicitação de exclusão gerada:

5.1.1. As variáveis do corpo da regra de exclusão que estão ligadas à cabeça da regra de derivação, são instanciadas com as constantes da tupla a ser excluída na relação derivada.

5.1.2. As variáveis do corpo da regra de exclusão em condições de igualdade com constantes ou com variáveis já instanciadas, recebem os valores associados.

5.1.3. Repete-se o passo 5.1.2 até que nenhuma nova variável possa ser instanciada.

5.2. Para cada solicitação de inserção gerada em função de literais negativos, são executados os passos 4.1.1 a 4.1.3.

5.3. Cada solicitação de exclusão é executada.

5.4. Cada solicitação de inserção gerada em função de literais negativos é executada.

O exemplo 5.18 demonstra o processo de realização de atualizações sobre uma relação derivada.

**Exemplo 5.18:** Considerando a seguinte relação derivada recursiva:

*ancestrais(Ancestral:X,Descendente:Y) :- pacientes(Nome:Y,Pai:X).*

*ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Z) ,*
*ancestrais(Ancestral:Z,Descendente:Y), X<>Y.*

As seguintes solicitações de inserção e exclusão seriam geradas:

*ins pacientes(Nome:Y,Pai:X).*

*del pacientes(Nome:Y,Pai:X) :- pacientes(Nome:Y,Pai:X).*

*del ancestrais(Ancestral:X,Descendente:Z):-*
*ancestrais(Ancestral:X,Descendente:Z),*
*ancestrais(Ancestral:Z,Descendente:Y),X<>Y.*

*del ancestrais(Ancestral:Z,Descendente:Y):-*
*ancestrais(Ancestral:X,Descendente:Z),*
*ancestrais(Ancestral:Z,Descendente:Y),X<>Y.*

As duas última regras de exclusão são geradas desta forma porque o literal *ancestrais* não é restrito (em relações derivadas, a chave é o conjunto de todos os seus atributos).

Considerando que o conteúdo das relações *pacientes* e *ancestrais* é:

*pacientes('João','Pedro').*
*pacientes('Pedro','Carlos').*

*ancestrais('Pedro', 'João').*
*ancestrais('Carlos','Pedro').*
*ancestrais('Carlos','João').*

Se a seguinte inserção fosse solicitada:

*ins ancestrais(Ancestral:'Paulo',Descendente:'Carlos').*

A única tupla que deve ser inserida é:

*ancestrais('Paulo','Carlos').*

Como a tupla não se encontra na relação *ancestrais*, as variáveis da regra de inserção são instanciadas:

*X = 'Carlos'* *Y = 'Paulo'*

A regra de inserção, com as variáveis instanciadas, pode ser aplicada:

*ins pacientes(Nome:'Carlos',Pai:'Paulo').*

As relações *pacientes* e *ancestrais* passam a ter as seguintes tuplas:

*pacientes('João','Pedro').*
*pacientes('Pedro','Carlos').*
*pacientes('Carlos','Paulo').*

*ancestrais('Pedro', 'João').*
*ancestrais('Carlos','Pedro').*
*ancestrais('Carlos','João').*
*ancestrais('Paulo','Carlos').*
*ancestrais('Paulo','Pedro').*
*ancestrais('Paulo','João').*

Se a seguinte exclusão fosse solicitada:

*del ancestrais('Paulo','João').*

Instanciando as variáveis da primeira regra de exclusão, teria-se:

*del pacientes(Nome:'João',Pai:'Paulo') :- pacientes(Nome:'João',Pai:'Paulo').*

Como não existe a tupla *pacientes('João','Paulo')*, a exclusão não é realizada. Instanciando as variáveis da segunda regra de exclusão teria-se:

*del ancestrais(Ancestral:'Paulo',Descendente:Z) :-*
*ancestrais(Ancestral:'Paulo',Descendente:Z),*
*ancestrais(Ancestral:Z,Descendente:'João'),'Paulo'<>'João'.*

O conjunto de tuplas que devem ser excluídas a partir desta regra é:
*ancestrais('Paulo','Carlos').*
*ancestrais('Paulo','Pedro').*

Para a tupla *ancestrais('Paulo','Carlos')*, instanciando as variáveis da primeira regra de exclusão, teria-se:

*del pacientes(Nome:'Carlos',Pai:'Paulo'):-*
*pacientes(Nome:'Carlos',Pai:'Paulo').*

Como existe esta tupla na relação *pacientes*, a exclusão é executada e o novo conteúdo das relações *pacientes* e *ancestrais* é:

*pacientes('João','Pedro').*
*pacientes('Pedro','Carlos').*

*ancestrais('Pedro', 'João').*
*ancestrais('Carlos','Pedro').*
*ancestrais('Carlos','João').*

Como a tupla *ancestrais('Paulo','Pedro')* não pode mais ser derivada, sua exclusão não pode ser realizada e o processo é encerrado.

O exemplo 5.18 demonstra que atualizações sobre relações derivadas definidas sobre outras relações derivadas, não necessariamente recursivas, pode levar a uma série de outras atualizações.

## 5.4 Comentários finais

O problema das atualizações sobre relações derivadas em Bancos de Dados Dedutivos é de difícil solução. No sistema DEDALO, são fornecidos mecanismos que buscam permitir que qualquer relação derivada seja atualizável. No entanto, a decisão de permitir ou não a atualização sobre uma relação derivada, deve ser cuidadosamente tomada. Atualizações não determinísticas nem sempre terão traduções que refletem a verdadeira intenção do usuário ao solicitar a atualização sobre a relação derivada.

# 6 Restrições de integridade

Restrições de integridade são condições que um Banco de Dados deve satisfazer em qualquer estado. Tipicamente, apenas uma limitada classe de restrições de integridade, como restrições sobre chaves e integridade referencial, são tratadas diretamente pelo Sistema Gerenciador de Banco de Dados. As restrições restantes devem ser gerenciadas pelos programas de aplicação.

No sistema DEDALO, as restrições de integridade são especificadas por regras de derivação. A regra que representa a restrição deve expressar uma situação que não deve ocorrer em nenhum estado válido do Banco de Dados. A cabeça da regra é o nome da restrição e os atributos da cabeça são as variáveis não ligadas a constantes do corpo. O exemplo 6.1 apresenta uma regra utilizada para definir uma restrição de integridade.

**Exemplo 6.1:** A seguinte restrição de integridade diz que nenhum paciente do sexo masculino pode ficar grávido:

*PacientesGrávidos(Nome:X) :- pacientes(Nome:X,Sexo:'M',Gravidez:'Sim').*

A vantagem deste tipo de especificação de restrições de integridade, além de permitir especificações declarativas das restrições, é que a restrição de integridade pode ser vista como uma relação derivada que deve permanecer sempre vazia, pois expressa uma situação que não deve ocorrer. Assim, se a aplicação da regra sobre o Banco de Dados derivar alguma tupla para a relação que representa a restrição, a restrição foi violada.

No exemplo 6.1, se a aplicação da regra sobre o Banco de Dados derivar alguma tupla para a relação *PacientesGrávidos*, então existem tuplas em *pacientes* que violam a restrição de integridade. Se a aplicação da regra resultar em uma relação vazia, então a restrição é satisfeita.

Obviamente, aplicar todas as regras que representam restrições de integridade sobre o Banco de Dados a cada modificação, não é um método eficiente para detectar violações de restrições de integridade [BRY 88]. Partindo do princípio de que o Banco de Dados era consistente no estado anterior e como a restrição pode ser vista como uma relação derivada que deve permanecer vazia, bastaria analisar as modificações ocorridas durante a transação e verificar se estas modificações induzem a inserção de uma tupla na relação derivada que representa a restrição de integridade. Se a inserção de uma tupla for detectada, a restrição foi violada pela transação. Vários métodos

eficientes de detecção de violações de integridade para Bancos de Dados Dedutivos podem ser encontrados na literatura [BRY 88, KOW 87, MOE 88, MOE 91, OLI 91].

Quando uma restrição de integridade é violada, a resposta tradicional de um Sistema de Banco de Dados, é abortar a transação. No entanto, existem métodos de reparar uma transação, modificando-a ou acrescentando operações a ela, de forma que a restrição de integridade não seja violada. Assim, se uma transação *T* leva o Banco de Dados a um estado inconsistente, o sistema criaria uma transação *T'* que é consistente e leva o Banco de Dados a um estado válido. A diferença entre *T'* e *T* é chamada de *reparo da transação* [BAE 94, CER 90a, FRA 93]. O objetivo do reparo é forçar a não violação da restrição de integridade (*integrity enforcement*).

Como uma restrição de integridade definida por uma regra de derivação pode ser vista como uma relação derivada que deve permanecer vazia, forçar a não violação da restrição de integridade significa excluir tuplas da relação que representa a restrição. Esta exclusão pode ser tratada exatamente como uma exclusão sobre relações derivadas, discutida na seção **5**.2.2 do capítulo 5. No sistema DEDALO, a escolha entre desfazer uma transação em caso de violação de restrição e reparar a restrição violada, é feita, para cada restrição de integridade, pelo projetista da regra de restrição.

Na seção 6.1, o método utilizado para detectar possíveis violações de restrições de integridade, é apresentado. Na seção 6.2, a forma como as restrições violadas são reparadas no sistema DEDALO é descrita.

## 6.1 Detecção de violações de restrições de integridade

A forma como as restrições de integridade são definidas no sistema DEDALO permite que elas sejam vistas como relações derivadas que devem permanecer vazias. Caso uma transação insira uma tupla em uma relação que representa uma restrição de integridade, houve violação da restrição. Como aplicar todas as regras de derivação que definem as restrições sobre o estado do Banco de Dados ao fim da transação não é uma forma eficiente de detectar violações de integridade, deve-se analisar o conjunto de operações que formam a transação e decidir se este conjunto de operações gera tuplas para a relação que representa a restrição.

Os métodos apresentados no capítulo 4 para propagação de atualizações para relações derivadas materializadas, detectam, a partir de um conjunto de operações de atualização, operações de exclusão e inserção que devem ser aplicadas sobre uma relação derivada materializada. Os mesmos métodos podem ser utilizados para detecção de violações de restrições de integridade da seguinte forma: todas as operações de exclusão e inserção, geradas pelo método de propagação aplicado ao conjunto de operações de atualização da transação, são aplicadas sobre a relação que representa a restrição. Se ao final da aplicação destas operações, a relação que representa a restrição não estiver vazia, a transação violou a restrição. O exemplo 6.2

demonstra a utilização das técnicas de propagação de atualizações para relações derivadas materializadas como técnicas de detecção de violação de restrições de integridade.

**Exemplo 6.2:** Considerando uma relação básica *pacientes*, cujo esquema é *Pacientes(Nome,Médico_responsável,Setor)* e cuja chave é *Nome*, e, considerando que exista a seguinte restrição de integridade, que determina que um médico não pode ser responsável por pacientes que estão em setores diferentes:

*MedResp(Médico:M,Setor:S) :-*
*pacientes(Médico_responsável:M,Setor:S) ,*
*pacientes(Médico_responsável:M,Setor:S2) , S <> S2.*

Como *Nome* é a chave para *pacientes*, ambos os literais seriam não seguros. O método DRed, geraria as seguintes solicitações de inserção e exclusão para a regra *MedResp*:

Exclusão:

*del MedResp(Médico:M,Setor:S):-MedResp(Médico:M,Setor:S),*
*del_pacientes(Médico_responsável:M,Setor:S).*

*del MedResp(Médico:M,Setor:S):- MedResp(Médico:M,Setor:S),*
*del_pacientes(Médico_responsável:M,Setor:S2).*

Re-inserção:

*ins MedResp(Médico:M,Setor:S) :- del_MedResp(Médico:M,Setor:S) ,*
*pacientes(Médico_responsável:M,Setor:S) ,*
*pacientes(Médico_responsável:M,Setor:S2) , S <> S2.*

Inserção:

*ins MedResp(Médico:M,Setor:S) :-*
*ins_pacientes(Médico_responsável:M,Setor:S),*
*pacientes(Médico_responsável:M,Setor:S2) , S <> S2.*

*ins MedResp(Médico:M,Setor:S) :-*
*pacientes(Médico_responsável:M,Setor:S),*
*ins_pacientes(Médico_responsável:M,Setor:S2),S <> S2.*

Supondo que o conteúdo da relação *pacientes* antes da transação seja:

*pacientes('Pedro','João','UTI').*
*pacientes('Maria','Ana','Maternidade').*
*pacientes('Carlos','João','UTI').*

Se a transação inserisse a seguinte tupla:

*pacientes('Paulo','Ana','UTI').*

As solicitações de exclusão e a de re-inserção não causariam nenhuma modificação na relação *MedResp*. No entanto, as solicitações de inserção, inseririam as

seguintes tuplas na relação *MedResp*:

*MedResp('Ana','UTI').*
*MedResp('Ana','Maternidade').*

Como a relação *MedResp* não permaneceu vazia após a aplicação das solicitações de atualização geradas pelo método DRed, a restrição de integridade foi violada.

Assim como na propagação de atualizações para relações derivadas, a utilização destes métodos na detecção de violações de restrições de integridade, é mais eficiente do que a aplicação das regras de restrição de integridade sobre o Banco de Dados a cada modificação.

## 6.2 Reparos de restrições de integridade

Se uma transação viola uma restrição de integridade, o sistema pode desfazer a transação ou gerar um conjunto de operações que reparem a restrição de integridade, de forma que, com estas operações adicionais, a transação leve o Banco de Dados a um estado consistente.

Como as regras que definem as restrições de integridade são vistas como relações derivadas que devem permanecer vazias, se ao final da transação existirem tuplas na relação que representa a restrição, houve violação de integridade. Para reparar a transação, o sistema deve gerar solicitações de exclusão das tuplas que permaneceram na relação que representa a restrição. Isto equivale a uma solicitação de exclusão sobre uma relação derivada.

As exclusões sobre relações derivadas foram discutidas no capítulo 5. A forma utilizada no sistema DEDALO para reparar restrições violadas, é gerar uma solicitação de exclusão para cada tupla da relação derivada que representa a restrição. Estas solicitações serão traduzidas em solicitações de atualização sobre relações básicas, utilizando o procedimento descrito no capítulo 5.

Para a violação de integridade do exemplo **6**.2, as seguintes solicitações de exclusão seriam geradas:

(1) *del MedResp(Médico:'Ana',Setor:'UTI').*
(2) *del MedResp(Médico:'Ana',Setor:'Maternidade').*

A solicitação 1 seria traduzida na seguinte exclusão sobre a relação *pacientes*:

*del pacientes(Médico_responsável:'Ana',Setor:'UTI') :-*
*pacientes(Médico_responsável:'Ana',Setor:'UTI'),*
*pacientes(Médico_responsável:'Ana',Setor:S2) , 'UTI' <> S2.*

A execução desta solicitação excluiria a tupla *pacientes('Paulo','Ana','UTI').* Como, a partir desta exclusão, a tupla *MedResp('Ana','Maternidade')* não será mais

derivada, a solicitação de exclusão 2 não será realizada. A relação *MedResp* estará vazia, o que indica que a restrição não foi violada.

Os reparos de restrições de integridade são especialmente úteis nas situações em que se deseja forçar a satisfação de uma restrição através de modificações no Banco de Dados. Este tipo de ação é tipicamente executada por Sistemas Gerenciadores de Banco de Dados para manter a integridade referencial. Quando a integridade referencial é violada e algumas tuplas se tornam órfãs, o sistema pode abortar a transação, desfazendo a atualização que violou a integridade referencial, ou executar a exclusão das tuplas órfãs. Esta última opção é uma ação de reparo. O exemplo 6.3 demonstra uma restrição de integridade onde a execução de uma ação de reparo é mais adequada do que abortar a transação em caso de violação.

**Exemplo 6.3:** Supondo que cada médico que atenda um paciente com tuberculose receba uma gratificação adicional. Os médicos que preenchem este requisito são relacionados na relação *gratificação.* As seguintes regras definem uma restrição de integridade que determina que não podem existir médicos atendendo pacientes com tuberculose sem gratificação, nem médicos recebendo gratificações sem atender pacientes com tuberculose:

(1) *NãoGratificados(Médico:X) :-*
*pacientes(Médico_Responsável:X,Diagnóstico:'tuberculose') ,*
*not(gratificação(Médico:X)).*

(2) *NãoGratificados(Médico:X) :-*
*gratificação(Médico:X) ,*
*not(pacientes(Médico_Responsável:X,Diagnóstico:'tuberculose')).*

Para a regra 1, as seguintes traduções da operação de exclusão seriam geradas:

(1.1) *del pacientes(Médico_Responsável:X,Diagnóstico:'tuberculose') :-*
*pacientes(Médico_Responsável:X,Diagnóstico:'tuberculose') ,*
*not(gratificação(Médico:X)).*

(1.2) *ins gratificação(Médico:X).*

Para a regra 2, a seguinte tradução da operação de exclusão seria gerada:

(2.1) *del gratificação(Médico:X) :- gratificação(Médico:X).*

Para a regra 1, o projetista deve escolher uma das atualizações geradas. A atualização escolhida seria a 1.2, pois é mais correto inserir o médico na relação de gratificações do que excluir o paciente com tuberculose.

Supondo que a relação *pacientes* e a relação *gratificação* estejam vazias e a seguinte inserção é realizada durante a transação:

*ins pacientes(Nome:'Maria', Médico_responsável:'Pedro',*
*diagnóstico:'tuberculose').*

A restrição seria violada, pois a tupla *NãoGratificados(Médico:'Pedro')* seria derivada. Para reparar esta transação, a seguinte solicitação de exclusão é gerada:

*del NãoGratificados(Médico:'Pedro').*

Aplicando a tradução 1.2, teria-se:

*ins gratificação(Médico:'Pedro').*

A restrição não é mais violada e a relação que armazena as gratificações está correta.

Se o pacientes incluído anteriormente for excluído, a restrição é novamente violada, pois, pela regra 2, a tupla *NãoGratificados(Médico:'Pedro')* pode ser derivada. Para reparar esta restrição, novamente a seguinte solicitação de exclusão é gerada:

*del NãoGratificados(Médico:'Pedro').*

A tradução 1.2 não pode ser aplicada porque a tupla *gratificação(Médico:'Pedro')* já se encontra na relação *gratificação*. Aplicando a tradução 2.1, teria-se:

*del gratificação(Médico:'Pedro') :- gratificação(Médico:'Pedro').*

A tupla *gratificação(Médico:'Pedro')* seria excluída e a restrição voltaria a ser satisfeita, mantendo, outra vez, a relação *gratificação* atualizada.

## 6.3 Comentários finais

O problema do não determinismo em atualizações sobre relações derivadas, não é necessariamente um problema com respeito a reparos de transações. O objetivo do reparo é encontrar um conjunto de operações sobre o Banco de Dados que, adicionado à transação original, levem o BD a um estado consistente. A existência de várias soluções possíveis para o problema, portanto, não é ruim. De fato, o problema das atualizações sobre visões é considerada uma das raízes culturais de *integrity enforcement* [FRA 93].

No sistema DEDALO, o módulo Gerenciador de Regras é utilizado para a especificação das regras de restrições de integridade. Para cada restrição definida, o projetista da restrição deve informar que ação deve ser tomada em caso de violação da restrição. As ações possíveis são abortar a transação ou gerar um conjunto de reparos para a restrição violada. Se a escolha for por reparar a transação, o sistema irá gerar as solicitações de exclusão ( e de inserção, se houverem literais negativos na regra ), que irão tornar a transação correta.

Caso a regra que define a restrição de integridade possua literais negativos, a tradução irá gerar solicitações de inserção para estes literais. Se o literal não for restrito, como apresentado no capítulo 5, a inserção não poderá ser feita. Caso o

reparo possa ser feito através de solicitações de exclusão sobre outros literais da regra, não haverá problema. Se isto não puder ser feito, o sistema será incapaz de gerar um reparo. O cuidado em criar uma regra de restrição de integridade que possa ser reparada é obrigação do projetista da regra.

A detecção da violação das restrições de integridade é realizada após a propagação das atualizações para relações derivadas materializadas.

# 7 Gerenciamento das transações

A execução de um conjunto de operações de atualização sobre o Banco de Dados Dedutivo pode desencadear uma série de outras operações. Se for solicitada uma alteração sobre uma relação derivada, esta será traduzida em um conjunto de novas operações sobre outras relações básicas ou derivadas. Se a atualização de uma relação básica provoca alterações em uma relação derivada materializada, uma série de operações, que propagarão as atualizações para a relação derivada materializada, serão geradas. Se uma restrição de integridade for violada e se deseja repará-la, operações de reparo serão executadas. Estas operações, por sua vez, poderão dar origem a novas propagações e alterações sobre relações derivadas. O conjunto de todas as operações de atualização sobre o Banco de Dados, desencadeadas pela solicitação de atualização original, formam a transação do Banco de Dados Dedutivo [MON 93].

As operações que formam uma transação em um Banco de Dados Dedutivo podem ser divididas em:

? Operações sobre relações básicas, que podem ter sido solicitadas pela transação original ou podem ter sido geradas em função de reparos de restrições de integridade ou de traduções de atualizações sobre relações derivadas;

? Operações sobre relações derivadas, que podem ter sido solicitadas pela transação original ou podem ter sido geradas em função de reparos de restrições de integridade, propagações de atualizações para relações derivadas materializadas ou de traduções de atualizações sobre relações derivadas.

As fontes de operações sobre relações básicas ou derivadas durante uma transação, podem ser divididas em:

? Transação original, que é o conjunto de operações sobre relações básicas e derivadas solicitadas pelo usuário originalmente;

? Traduções de atualizações sobre relações derivadas, que, como discutido no capítulo 5, geram uma série de outras operações sobre relações básicas ou sobre relações derivadas, caso a relação que se deseja atualizar seja definida em termos de outras relações derivadas;

? Propagação de atualizações para relações derivadas materializadas, que geram um conjunto de operações sobre as relações derivadas que estão materializadas;

? Reparos de restrições de integridade, que geram operações sobre relações básicas e derivadas, com o objetivo de reparar restrições de integridade violadas.

Cada uma das operações, geradas em cada uma das fontes, podem dar origem a novas operações. Assim, por exemplo, se uma restrição de integridade está definida sobre uma relação derivada materializada e se as operações geradas pela propagação das atualizações violam a restrição, podem ser gerados reparos que darão origem a novas operações que, por sua vez, podem gerar novas propagações e assim sucessivamente.

O processamento de uma transação pode ser dividido em cinco fases:

1. Execução da transação original, onde são executadas as solicitações de exclusão e inserção solicitadas pelo usuário;

2. Execução das traduções de atualizações sobre relações derivadas, onde as alterações sobre relações básicas que tornam visíveis as atualizações solicitadas sobre as relações derivadas são executadas;

3. Propagação das atualizações para relações derivadas materializadas;

4. Verificação das restrições de integridade, onde são detectadas possíveis violações;

5. Execução de reparos das restrições de integridade violadas.

Cada uma destas fases será analisada na seção 7.1. A seção 7.2 apresentará detalhes sobre a execução de transações no sistema DEDALO.

## 7.1 As fases da transação

### 7.1.1 A execução da transação original

Na transação original, um conjunto de operações, sobre relações básicas ou derivadas, é solicitado. As operações sobre relações básicas podem ser executadas diretamente, mas as atualizações sobre relações derivadas devem ser traduzidas em termos de atualizações sobre relações básicas. Esta tradução irá gerar um novo conjunto de operações. A execução destas operações, que pode ser vista como uma subtransação, é realizada na fase 2, descrita na seção 7.1.2.

Quando a transação original atinge uma operação sobre uma relação derivada, a transação original permanece suspensa, até que o conjunto de operações que torna visível a atualização sobre a relação derivada, seja executado, como mostra a figura 7.1.

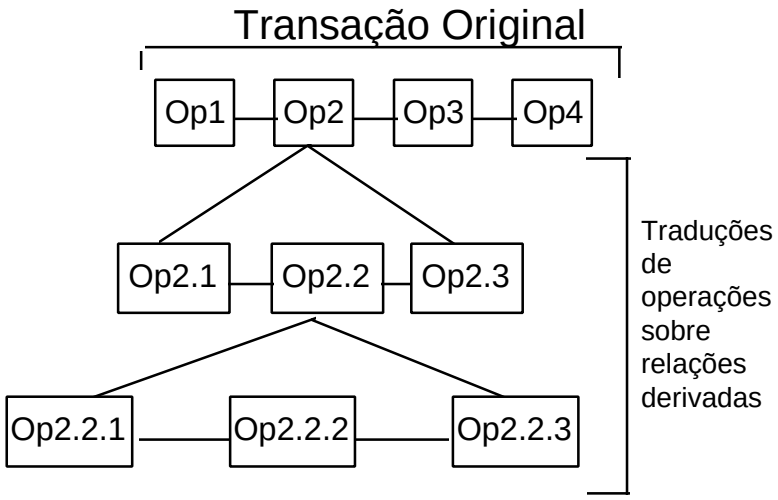


FIGURA 7.1 - Exemplo de execução da transação original

Na figura 7.1, a transação original é composta pelas operações *op1, op2, op3* e *op4*. As operações *op1, op3* e *op4* se referem a relações básicas. A operação *op2* é uma operação sobre uma relação derivada. A tradução desta operação gerou três novas operações *op2.1, op2.2* e *op2.3*. Como a operação *op2.2* também se refere a uma relação derivada, ela é traduzida nas operações *op2.2.1, op2.2.2* e *op2.2.3*. A transação original fica suspensa até que a operação *op2* seja realizada e isto equivale a aguardar que toda a subárvore de execução da tradução da operação seja executada. Quando a execução da operação *op2* acaba, a transação original pode continuar.

Se a subtransação que executa a tradução da operação derivada for abortada, a relação original também é abortada.

### 7.1.2 A execução das traduções de operações sobre relações derivadas

Quando uma operação sobre uma relação derivada é solicitada, ela é traduzida em um novo conjunto de operações que tornarão a atualização solicitada visível na relação derivada. Sempre que a tradução de uma operação está sendo executada, a subtransação que contém a operação ficará suspensa. Caso a subtransação seja abortada, a transação inteira também é abortada.

Como foi apresentado na figura 7.1, da seção anterior, a execução das traduções de operações sobre relações derivadas é feita através da criação de uma subtransação que executa as operações de tradução. Caso uma destas operações se refira a uma relação derivada, uma nova subtransação é criada. Quando todas as subtransações são executadas, a operação sobre a relação derivada está resolvida e a transação que a solicitou pode continuar.

### 7.1.3 A execução das propagações para relações derivadas materializadas

Quando a transação original acaba, inicia-se a fase de execução das propagações de atualizações para relações derivadas materializadas. Como as relações derivadas materializadas podem ser definidas em termos de outras relações derivadas, estas relações são ordenadas em níveis, de forma que, no primeiro nível, estão todas as relações definidas somente em termos de relações básicas, e no nível *n* estão todas as relações definidas em termos de relações básicas e relações derivadas dos níveis *1* a *n-1*.

A ordenação das relações derivadas em níveis, permite que se trate as relações derivadas dos níveis anteriores exatamente como relações básicas, pois já estarão totalmente computadas. A única exceção são as relações recursivas. O método de propagação para este tipo de relação, no entanto, através de um tratamento especial, é capaz de propagar corretamente as atualizações.

Como mostra a figura 7.2, a propagação gerada a partir das operações da transação original e das operações geradas pelas traduções das operações sobre relações derivadas, possui três operações, *op5, op6* e *op7*.

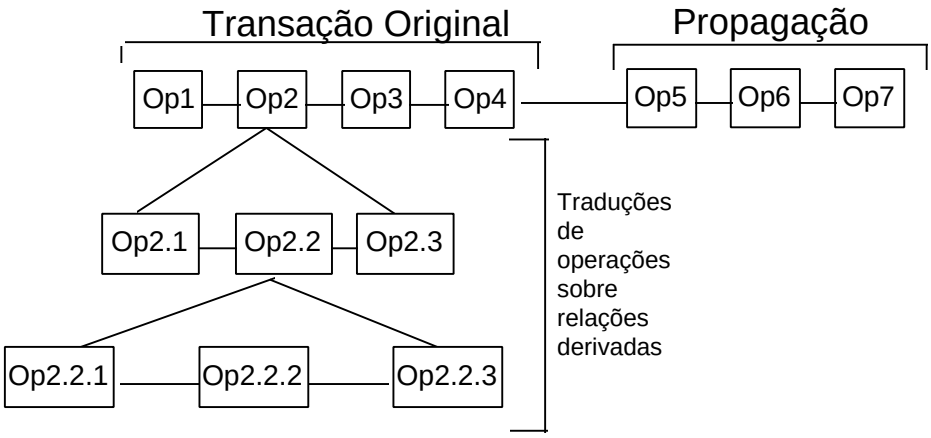


FIGURA 7.2 - As operações de propagação e a transação original

### 7.1.4 A verificação das restrições de integridade

Na fase de verificação das restrições de integridade, as modificações provocadas no Banco de Dados pela transação até o momento, são analisadas e as possíveis violações de integridade são detectadas. Se nenhuma violação é encontrada, a transação levou o Banco de Dados a um estado válido e pode terminar, caso contrário, a transação é abortada ou reparada, de acordo com as opções do projetista no momento da definição das restrições de integridade.

A fase de verificação das restrições de integridade também gera operações sobre o Banco de Dados. Estas operações são as solicitações de inserção sobre as relações que representam as restrições de integridade. Caso alguma restrição de integridade seja violada, estas operações também farão parte da transação.

### 7.1.5 A execução dos reparos das restrições de integridade

Se uma restrição de integridade é violada, o projetista pode escolher entre abortar a transação ou repará-la. Os reparos das restrições de integridade são solicitações de exclusão sobre as relações que representam as restrições de integridade. Estas operações serão traduzidas em outras operações sobre as relações básicas, que darão origem a uma nova fase de propagação e verificação de integridade.

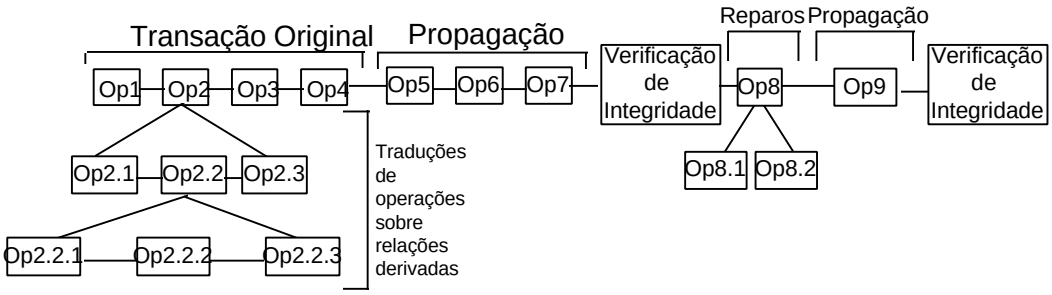


FIGURA 7.3 - Uma transação completa

A figura 7.3 apresenta a transação completa. Depois da execução dos reparos, uma nova fase de propagação é executada para propagar as modificações causadas pelos reparos. Ao final da propagação, uma nova verificação de integridade é realizada e, caso nenhuma restrição seja violada, a transação pode terminar.

## 7.2 A execução das transações no Sistema DEDALO

Quando uma transação é submetida ao Sistema DEDALO, cada uma das fases descritas da seção 7.1 é executada. Uma transação é um conjunto de operações de inserção ou exclusão sobre relações básicas ou derivadas do Banco de Dados Dedutivo.

O processamento da transação é feito da seguinte forma:

1. Um conjunto de operações $O_{to}$, que forma a transação original, solicitada por um usuário do Banco de Dados, é submetida ao sistema.

2. Para cada operação:

2.1. Se a operação se refere a uma relação básica, é imediatamente executada e a tupla inserida ou excluída é acrescentada na relação delta de inserção ou exclusão, correspondente àquela relação. As relações delta armazenam as tuplas inseridas e excluídas durante a transação.

2.2. Se a operação se refere a uma relação derivada, ela é traduzida em termos de operações sobre as relações que a derivam, e estas operações, que compõem o conjunto de operações $O_d$, são executadas, de acordo com os passos 2.1 e 2.2.

3. Quando as operações de $O_{to}$ foram todas executadas, inicia-se a propagação das atualizações sobre as relações derivadas materializadas. O conjunto de operações $O_m$ contém as operações de inserção e exclusão sobre as relações derivadas materializadas geradas pela propagação. As tuplas inseridas e excluídas das relações materializadas são acrescentadas às suas respectivas relações delta de inserção e exclusão.

4. A fase de verificação de restrições de integridade procura por violações de integridade, verificando se é possível derivar alguma tupla para as relações que representam as restrições. Cada tupla que é inserida/excluída neste processo, é também acrescentada às relações delta de inserção e exclusão referentes à relação que representa a restrição.

5. Se todas as relações derivadas que representam restrições de integridade permanecerem vazias depois do passo 4, a transação acaba.

6. Se alguma restrição de integridade, em cuja definição o projetista optou por não gerar reparos, foi violada, a transação é abortada.

7. Se foram definidas ações de reparos para todas as restrições de integridade violadas pela transação, as operações geradas por estes reparos, que compõem o conjunto $O_r$, são executadas, seguindo os passos 2 a 7.

A transação completa é formada pelo conjunto de operações da transação original, pelo conjunto de operações de tradução das atualizações sobre relações derivadas, pelo conjunto de operações de propagação e pelo conjunto de operações de reparo.

## 7.3 Comentários finais

Existem vários modelos de execução de transações em sistemas de Bancos de Dados que utilizam regras [DAY 94, TAN 95], ou seja, ativos e dedutivos. Em particular, cada Sistema de Banco de Dados Dedutivo ou Ativo, utiliza um modelo de execução que se adapta bem ao seu sistema. As características do sistema DEDALO (propagação, atualizações sobre derivadas, reparos de restrições), exigiram um processo de execução das transações diferenciado, onde cada fase é responsável por um conjunto de operações. Cada fase da transação, utiliza as modificações provocadas

pelas fases anteriores. Ao final, a transação será consistente ou será abortada e suas operações desfeitas.

O processamento das transações no sistema DEDALO é disparado toda vez que uma atualização é solicitada ao sistema. Estas atualizações são solicitações de inserção e exclusão de tuplas individuais. Mesmo se a solicitação foi feita por meio de uma regra, as tuplas individuais que podem ser derivadas da aplicação daquela regra são geradas, e solicitações de inserção e exclusão são criadas para cada tupla. Se é solicitada a alteração de uma tupla, uma solicitação de exclusão da tupla antiga e uma solicitação de inserção da tupla alterada são geradas.

# 8 DEDALO: Um Sistema Gerenciador de Banco de Dados Dedutivo

O Sistema Gerenciador de Bancos de Dados Dedutivos DEDALO[3] é um conjunto de ferramentas que permitem a utilização de regras de derivação baseadas em lógica na dedução de novas informações a partir das já contidas no Banco de Dados. As principais características do Sistema DEDALO são:

? Utiliza uma linguagem de consulta declarativa, com maior poder expressivo que o Datalog e álgebra relacional, que suporta regras recursivas, negação, operações aritméticas, operações de comparação, funções agregadas e possui uma sintaxe que permite a referência aos atributos pelo seu nome e não por sua posição dentro da relação como é tradicional nas linguagens de consulta baseadas em lógica;

? Suporta raciocínio aproximado através de adaptações da lógica fuzzy para Bancos de Dados Dedutivos;

? Suporta relações derivadas materializadas e possui mecanismos eficientes para sua manutenção;

? Suporta atualizações sobre relações derivadas, buscando eliminar o não determinismo destas atualizações através de uma interação com o projetista da regra no momento da definição da relação derivada;

? Permite a especificação de restrições de integridade, utilizando a própria linguagem do sistema, possuindo métodos eficientes para detecção de violações;

? É capaz de gerar reparos para restrições de integridade violadas, o que permite corrigir uma transação sem abortá-la;

? Foi construído utilizando-se uma arquitetura cliente-servidor, onde as ferramentas DEDALO são processos clientes e o servidor é um Sistema de Banco de Dados relacional. A comunicação entre as ferramentas do sistema DEDALO e o Sistema Gerenciador do Banco de Dados é realizada via ODBC (Open Database Connectivity), o que permite a utilização do sistema DEDALO com qualquer Sistema Gerenciador de Banco de Dados Relacional que possua uma interface ODBC;

---

[3] Como na linguagem homônima, DEDALO = Dedução, Dados e Lógica

? O sistema DEDALO foi construído utilizando a ferramenta para construção de aplicações cliente/servidor Delphi ([BOR 95, BOR 95a, BOR 95b, RUB 95]), para a qual foram criados novos componentes que permitem ao usuário desenvolver suas aplicações sobre Sistemas de Banco de Dados que utilizam o DEDALO, através do próprio Delphi.

A seção 8.1 descreve a arquitetura do sistema DEDALO, apresentando cada um dos seus componentes e a seção 8.2 apresenta alguns pontos importantes sobre a implementação do sistema.

## 8.1 Arquitetura do sistema

O Sistema DEDALO foi construído em uma arquitetura cliente/servidor, onde as ferramentas DEDALO são executadas como processos clientes e qualquer Sistema de Banco de Dados Relacional, que possua interface com ODBC e seja compatível com a linguagem SQL/ANSI [AME 92], pode ser utilizado como servidor.

A comunicação entre as ferramentas do Sistema DEDALO e o Sistema Gerenciador de Banco de Dados é feita via ODBC (Open Database Connectivity). A interface ODBC, definida pela Microsoft Corporation, é uma interface padrão para sistemas de gerenciamento de Banco de Dados nos ambientes Windows e Windows NT. Os principais Sistemas de Banco de Dados Relacionais comerciais, como Oracle, Sybase, Watcom, Interbase e muitos outros, possuem interface para o ODBC. A utilização desta arquitetura pelo sistema DEDALO, permite que ele possa ser utilizado com uma grande variedade de Sistemas Gerenciadores de Bancos de Dados, permitindo, também, que Sistemas de BD convencionais possam ser utilizados como um Banco de Dados Dedutivo.

O Sistema DEDALO está dividido em quatro módulos: o *Gerenciador de Regras*, que é a ferramenta utilizada na criação e manutenção das regras de derivação, que darão origem a relações derivadas, e na definição e manutenção das regras de restrições de integridade; a *Interface Interativa*, que é uma ferramenta desenvolvida para processar solicitações de consultas *ad hoc* e solicitações de atualização sobre o Banco de Dados; um conjunto de *novos componentes Delphi*, que são novas classes criadas especificamente para a construção de aplicações sobre o sistema DEDALO, utilizando o ambiente de desenvolvimento de aplicações cliente/servidor do Delphi; e o *Tradutor de Sentenças DEDALO/SQL-ANSI*, que traduz a definição das regras, as solicitações de consulta e as solicitações de atualização feitas na linguagem DEDALO para comandos SQL-ANSI, que serão submetidos ao Sistema de Banco de Dados servidor. A figura 8.1 apresenta um diagrama da arquitetura do sistema DEDALO.

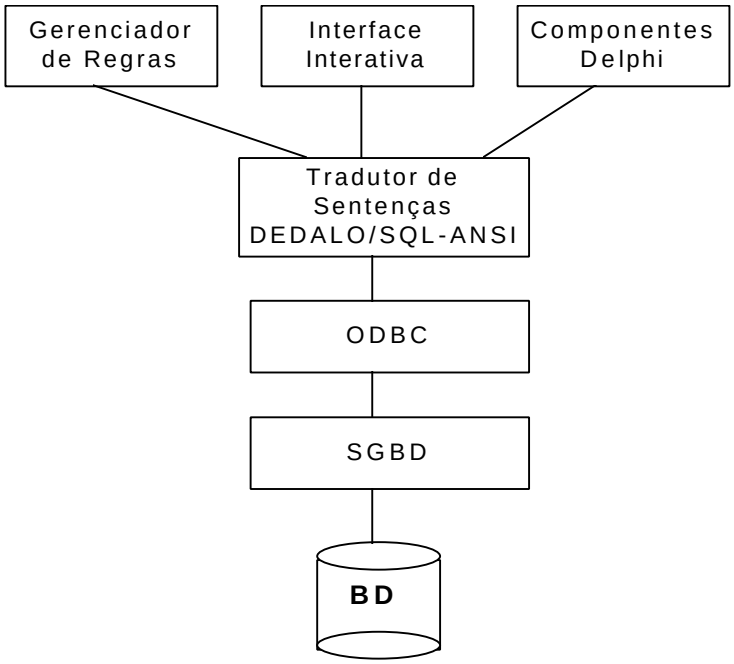


FIGURA 8.1 - O diagrama da arquitetura do sistema DEDALO

O Banco de Dados Relacional contendo as relações básicas, sobre as quais as regras de derivação e as regras de restrição de integridade serão definidas, deve ser criado através das ferramentas disponíveis no Sistema Gerenciador do Banco de Dados.

Nas próximas seções, cada uma das ferramentas que compõem o Sistema DEDALO será descrita.

### 8.1.1 O Gerenciador de Regras

O gerenciador de regras é a ferramenta utilizada na criação e manutenção das relações derivadas e das restrições de integridade do Banco de Dados. O processo de criação de regras de derivação e de regras de integridade será descrito a seguir.

#### 8.1.1.1 A definição das relações derivadas

A criação de uma relação derivada exige que o projetista forneça ao gerenciador de regras um conjunto de regras de derivação que irão definir a relação derivada. O gerenciador de regras, então, faz uma análise léxica e sintática das regras informadas, detectando possíveis erros.

Se o conjunto de regras que define a relação derivada está sintaticamente correto, o gerenciador irá aplicar uma série de regras de normalização [BÖH 94] sobre as regras de derivação informadas. As regras de normalização irão transformar as

regras de derivação, de forma que as regras transformadas não possuirão disjunção e poderão ser adequadamente processadas pelos algoritmos que geram as operações de propagação de atualizações para relações derivadas materializadas, pelos algoritmos que geram as traduções de atualizações sobre as relações derivadas e pelo algoritmo que gera a tradução das regras para sentenças SQL-ANSI. Um conjunto de regras é dito normalizado se nenhuma das seguintes regras de normalização puder ser aplicada:

| | | | |
|---|---|---|---|
| (R1) | *not(A , B)* | ? | *not(A) ; not(B)* |
| (R2) | *not(A ; B)* | ? | *not(A) , not(B)* |
| (R3) | *not(not(A))* | ? | *A* |
| (R4) | *not(E1 > E2)* | ? | *E1 <= E2* |
| | *not(E1 < E2)* | ? | *E1 >= E2* |
| | *not(E1<=E2)* | ? | *E1 > E2* |
| | *not(E1>=E2)* | ? | *E1 < E2* |
| | *not(E1 = E2)* | ? | *E1 <> E2* |
| | *not(E1<>E2)* | ? | *E1 = E2* |
| (R5) | *A , (B ; C)* | ? | *(A , B) ; (A , C)* |
| (R6) | *(A ; B) , C* | ? | *(A , C) ; (B , C)* |
| (R7) | *A :- B ; C* | ? | *A :- B* |
| | | | *A :- C* |

Depois da normalização, o gerenciador passa a verificar se cada uma das regras é segura, segundo a definição da seção 3.10, capítulo 3. Se todas as regras são seguras, a compatibilidade das cabeças é testada, buscando encontrar cabeças de regras com número de atributos diferentes, ou tipos de atributos diferentes, ou, ainda, solicitações de ordenação das tuplas derivadas incompatíveis entre si. Se as regras possuem cabeças compatíveis, o gerenciador irá verificar se as que possuem negação e funções agregadas são estratificadas. Se nenhum erro for detectado pelo gerenciador, o conjunto de regras de derivação é aceito.

Depois de definir um conjunto de regras válido, o projetista deve informar se a relação derivada será virtual ou materializada. Se for materializada, o gerenciador de regras irá criar um conjunto de operações de propagação das atualizações para a relação derivada materializada. A seguir, o projetista deverá informar se serão permitidas atualizações sobre a relação derivada. Em caso afirmativo, o sistema irá gerar um conjunto de operações de tradução das atualizações sobre a relação derivada. Este conjunto de operações será apresentado ao projetista, que deverá auxiliar o sistema a criar a tradução, decidindo por uma ou outra, em caso de não determinismo, e criando regras de tradução nos casos em que o sistema for incapaz de encontrar uma tradução adequada. Mesmo que uma tradução determinística e correta esteja disponível, o projetista pode optar por criar suas próprias regras de tradução.

Quando a relação derivada está definida, o gerenciador de regras colocará as regras no catálogo de regras. O catálogo de regras é um conjunto de relações que armazenam informações sobre as regras definidas. As relações que compõem o catálogo são:

? *SYSDERIVADAS*, onde são armazenadas as informações básicas das relações derivadas;

? *SYSREGRAS*, onde são armazenadas as regras originais não normalizadas usadas na definição das relações derivadas;

? *SYSCABEÇAS*, onde as cabeças das regras normalizadas são armazenadas;

? *SYSCORPOS*, onde os corpos das regras normalizadas são armazenados;

? *SYSARGUMENTOS*, onde os argumentos dos literais do corpo são armazenados.

Os esquemas das relações *SYSDERIVADAS*, *SYSREGRAS*, *SYSCABEÇAS*, *SYSCORPOS* e *SYSARGUMENTOS* são os seguintes:

*SYSDERIVADAS(Relação, Materializada, Atualizável,Recursiva)*

O atributo *Relação* armazena o nome da relação derivada, o atributo *Materializada* indica se a relação derivada será materializada ou não, *Atualizável* indica se poderão ser solicitadas atualizações sobre a relação derivada e *Recursiva* indica se a relação derivada é definida por um conjunto de regras recursivas ou não.

*SYSREGRAS(Número, Relação, Regra,CF)*

O atributo *Número* é um número de identificação da regra, *Relação* é a relação derivada que é definida pela regra, *Regra* é a regra de derivação informada e *CF* é o fator de certeza da regra.

*SYSCABEÇAS(Número, Relação, NoRegraNormalizada, Argumento, Atributo, Expressão, Anotação)*

Cada tupla da relação *SYSCABEÇAS* armazena informações sobre um atributo da cabeça de uma das regras normalizadas que formam a relação derivada. O atributo *Número* armazena o número de identificação da regra de derivação não normalizada, *Relação* é a relação derivada que é definida pela regra, *NoRegraNormalizada* é o número de identificação da regra normalizada, *Argumento* é um número seqüencial que indica a ordem do argumento dentro da cabeça, *Atributo* é o atributo da relação derivada definido pela tupla, *Expressão* é a expressão que representa o valor do atributo, podendo ser uma variável, uma constante, uma expressão aritmética ou uma função agregada e *Anotação* é um número inteiro indicando que existe uma ordenação das tuplas derivadas em função deste atributo.

*SYSCORPOS(Número, Relação, NoRegraNormalizada, NoLiteral, Literal, Negação)*

Cada tupla da relação *SYSCORPOS* armazena informações sobre um literal do corpo de cada regra normalizada que define a relação. O atributo *Número* armazena o número de identificação da regra de derivação não normalizada, *Relação* é a relação derivada que é definida pela regra, *NoRegraNormalizada* é o número de identificação da regra normalizada, *NoLiteral* é um número seqüencial que indica a ordem do literal

dentro do corpo, *Literal* é o nome da relação que é representada pelo literal e *Negação* indica se o literal é negativo ou positivo.

*SYSARGUMENTOS(Número, Relação, NoRegraNormalizada, NoLiteral, Argumento, Atributo, Expressão)*

Cada tupla da relação *SYSARGUMENTOS* armazena informações sobre um argumento de um literal do corpo da regra. O atributo *Número* armazena o número de identificação da regra de derivação não normalizada, *Relação* é a relação derivada que é definida pela regra, *NoRegraNormalizada* é o número de identificação da regra normalizada, *NoLiteral* é um número seqüencial que indica a ordem do literal dentro do corpo, *Argumento* é um número seqüencial que indica a ordem do argumento dentro do literal, *Atributo* é o atributo da relação correspondente ao literal definido pela tupla, *Expressão* é a expressão que representa o valor do atributo, podendo ser uma variável, uma constante ou uma expressão aritmética. O atributo *Expressão* pode armazenar também uma expressão de comparação, se o literal não corresponde a uma relação.

O exemplo 8.1 apresenta a representação de uma regra nas relações do catálogo.

**Exemplo 8.1:** Considerando a seguinte regra de derivação não normalizada, que define uma relação derivada materializada e não atualizável:

*PacientesNaUTI(Nome:X@1) :- pacientes(Nome:X, Setor:'UTI1') ;*
*pacientes(Nome:X,Setor:'UTI2').*

A versão normalizada da regra é:

*PacientesNaUTI(Nome:X@1) :- pacientes(Nome:X, Setor:'UTI1').*
*PacientesNaUTI(Nome:X@1) :- pacientes(Nome:X,Setor:'UTI2').*

As relações do catálogo de regras conteriam as tuplas:

*SYSDERIVADAS('PacientesNaUTI','Sim','Não','Não').*

*SYSREGRAS(1,'PacientesNaUTI','PacientesNaUTI(Nome:X@1) :-*
*pacientes(Nome:X,Setor:'UTI1') ; pacientes(Nome:X,Setor:'UTI2').',1).*

*SYSCABEÇAS(1,'PacientesNaUTI',1,1,'Nome','X',1).*
*SYSCABEÇAS(1,'PacientesNaUTI',2,1,'Nome','X',1).*

*SYSCORPOS(1,'PacientesNaUTI',1,1,'pacientes','+').*
*SYSCORPOS(1,'PacientesNaUTI',2,1,'pacientes','+').*

*SYSARGUMENTOS(1,'PacientesNaUTI',1,1,1,'Nome','X').*
*SYSARGUMENTOS(1,'PacientesNaUTI',1,1,2,'Setor','UTI1').*
*SYSARGUMENTOS(1,'PacientesNaUTI',2,1,1,'Nome','X').*
*SYSARGUMENTOS(1,'PacientesNaUTI',2,1,2,'Nome','UTI2').*

As solicitações de atualização que realizam a propagação das atualizações para as relações derivadas materializadas são armazenadas na relação

*SYSPROPAGAÇÃO* e as traduções para as atualizações sobre relações derivadas são armazenadas na relação *SYSTRADUÇÃO*. Os esquemas destas relações são os seguintes:

*SYSPROPAGAÇÃO(Relação, NoPropagação, Atualização).*

O atributo *Relação* armazena o nome da relação derivada materializada, *NoPropagação* é um número que identifica a solicitação de atualização para propagação e *Atualização* é a solicitação de atualização que realiza a propagação.

*SYSTRADUÇÃO(Relação, NoTradução, Tradução, Tipo).*

O atributo *Relação* armazena o nome da relação derivada atualizável, *NoTradução* é um número que identifica a solicitação de atualização da tradução, *Tradução* é a solicitação de atualização e *Tipo* indica se a solicitação traduz uma inserção ou uma exclusão.

#### 8.1.1.2 A definição das restrições de integridade

A definição de uma restrição de integridade exige que o projetista informe uma regra de integridade que deve expressar uma condição que não deve ocorrer em um estado válido do Banco de Dados. O gerenciador de regras, então, faz uma análise léxica e sintática da regra informada, detectando possíveis erros.

Como para as regras de derivação, o gerenciador de regras irá aplicar as regras de normalização sobre a regra que define a restrição e a seguir verificará se a regra é segura e possui negação e funções agregadas estratificadas. Se a restrição de integridade for aceita, o sistema aplicará a regra de restrição sobre o estado atual do Banco de Dados, procurando por possíveis violações. Isto garante que a premissa de que o Banco de Dados sempre é consistente antes de uma transação, será verdadeira. Se forem encontradas violações, o projetista poderá optar entre reformular a regra de restrição ou solicitar reparos no Banco de Dados, de forma que a restrição seja aceita.

Depois que a validade da restrição está garantida, o projetista deve informar que ação o sistema deverá tomar em caso de violação da restrição: abortar a transação que a violou ou aplicar um conjunto de operações de reparo. Se a última alternativa for escolhida, o sistema irá gerar um conjunto de operações de reparo, que será apresentado ao projetista. O projetista, então, deverá escolher uma entre as alternativas de reparo disponíveis.

Quando a restrição de integridade está definida, o gerenciador de regras armazenará as regras de restrições no catálogo de regras. As relações do catálogo que armazenam informações sobre as restrições de integridade são:

? *SYSRESTRIÇÕES*, onde são armazenadas as informações básicas das restrições de integridade;

? *SYSRIREGRAS*, onde são armazenadas as regras originais não normalizadas usadas na definição das restrições de integridade;

? *SYSRICABEÇAS*, onde as cabeças das regras de restrições de integridade normalizadas são armazenadas;

? *SYSRICORPOS*, onde os corpos das regras de restrição de integridade normalizadas são armazenados;

? *SYSRIARGUMENTOS*, onde os argumentos dos literais do corpo são armazenados.

Os esquemas das relações *SYSRESTRIÇÕES*, *SYSRIREGRAS*, *SYSRICABEÇAS*, *SYSRICORPOS* e *SYSRIARGUMENTOS* são os seguintes:

*SYSRESTRIÇÕES(Restrição, Reparar,Recursiva)*

O atributo *Restrição* armazena o nome da restrição de integridade, o atributo *Reparar* indica se a transação que violou a restrição de integridade deverá ser reparada ou abortada e o atributo *Recursiva* indica se a restrição de integridade é definida por um conjunto de regras recursivas ou não.

*SYSRIREGRAS(Número, Restrição, Regra)*

O atributo *Número* é um número de identificação da regra de restrição, *Restrição* é o nome da restrição de integridade definida pela regra e *Regra* é a regra de integridade informada.

*SYSRICABEÇAS(Número, Restrição, NoRegraNormalizada, Argumento, Atributo, Expressão)*

Cada tupla da relação *SYSRICABEÇAS* armazena informações sobre um atributo da cabeça de uma das regras normalizadas que definem a restrição de integridade. O atributo *Número* armazena o número de identificação da regra não normalizada que define a restrição, *Restrição* é o nome da restrição de integridade definida pela regra, *NoRegraNormalizada* é o número de identificação da regra normalizada, *Argumento* é um número seqüencial que indica a ordem do argumento dentro da cabeça, *Atributo* é o atributo da relação derivada que representa a restrição e *Expressão* é a expressão que representa o valor do atributo, podendo ser uma variável, uma constante, uma expressão aritmética ou uma função agregada.

*SYSRICORPOS(Número, Restrição, NoRegraNormalizada, NoLiteral, Literal, Negação)*

Cada tupla da relação *SYSRICORPOS* armazena informações sobre um literal do corpo de cada regra normalizada que define a restrição de integridade. O atributo *Número* armazena o número de identificação da regra de integridade não normalizada, *Restrição* é o nome da restrição de integridade definida pela regra, *NoRegraNormalizada* é o número de identificação da regra normalizada, *NoLiteral* é um número seqüencial que indica a ordem do literal dentro do corpo, *Literal* é o nome da relação que é representada pelo literal e *Negação* indica se o literal é negativo ou positivo.

*SYSRIARGUMENTOS(Número, Restrição, NoRegraNormalizada, NoLiteral, Argumento, Atributo, Expressão)*

Cada tupla da relação *SYSRIARGUMENTOS* armazena informações sobre um argumento de um literal do corpo da regra. O atributo *Número* armazena o número de identificação da regra de integridade não normalizada, *Restrição* é o nome da restrição de integridade definida pela regra, *NoRegraNormalizada* é o número de identificação da regra normalizada, *NoLiteral* é um número seqüencial que indica a ordem do literal dentro do corpo, *Argumento* é um número seqüencial que indica a ordem do argumento dentro do literal, *Atributo* é o atributo da relação correspondente ao literal definido pela tupla, *Expressão* é a expressão que representa o valor do atributo, podendo ser uma variável, uma constante ou uma expressão aritmética. O atributo *Expressão* pode armazenar também uma expressão de comparação, se o literal não corresponde a uma relação.

O exemplo 8.2 apresenta a representação de uma restrição de integridade nas relações do catálogo.

**Exemplo 8.2:** Considerando a seguinte regra não normalizada, que define uma restrição de integridade que não deve ser reparada em caso de violação:

*MedResp(Médico:M,Setor:S) :-*
*pacientes(Médico_responsável:M,Setor:S) ,*
*pacientes(Médico_responsável:M,Setor:S2) , not(S = S2).*

A versão normalizada da regra é:

*MedResp(Médico:M,Setor:S) :- pacientes(Médico_responsável:M,Setor:S) ,*
*pacientes(Médico_responsável:M,Setor:S2) , S <> S2.*

As relações do catálogo de regras conteriam as tuplas:

*SYSRESTRIÇÕES('MedResp','Não','Não').*

*SYSRIREGRAS(1,'MedResp','MedResp(Médico:M,Setor:S) :-*
*pacientes(Médico_responsável:M,Setor:S) ,*
*pacientes(Médico_responsável:M,Setor:S2) , not(S = S2).').*

*SYSRICABEÇAS(1,'MedResp',1,1,'Médico','M').*
*SYSRICABEÇAS(1,'MedResp',1,2,'Setor','S').*

*SYSRICORPOS(1,'MedResp',1,1,'pacientes','+').*
*SYSRICORPOS(1,'MedResp',1,2,'pacientes','+').*
*SYSRICORPOS(1,'MedResp',1,3,'','+').*

*SYSRIARGUMENTOS(1,'MedResp',1,1,1,'Médico_responsável','M').*
*SYSRIARGUMENTOS(1,'MedResp',1,1,2,'Setor','S').*
*SYSRIARGUMENTOS(1,'MedResp',1,2,1,'Médico_responsável','M').*
*SYSRIARGUMENTOS(1,'MedResp',1,2,2,'Setor','S2').*
*SYSRIARGUMENTOS(1,'MedResp',1,3,1,'','S <> S2').*

A relação *SYSREPAROS* armazena as ações de reparos das restrições violadas. O esquema desta relação é:

*SYSREPAROS(Restrição, NoReparo, Reparo).*

O atributo *Restrição* armazena o nome da restrição de integridade a que o reparo se refere, *NoReparo* é um número que identifica a operação de reparo e *Reparo* é a solicitação de atualização que repara a transação.

### 8.1.2 A Interface Interativa

A interface interativa permite que o usuário submeta interativamente comandos da linguagem DEDALO a partir de um cliente sobre um Banco de Dados.

A interface interativa é composta por uma área onde as sentenças DEDALO são digitadas e uma área onde as respostas às sentenças são apresentadas. As sentenças digitadas, depois de analisadas e estando corretas, são transformadas em SQL pelo Tradutor de Sentenças DEDALO/SQL-ANSI, e então submetidas ao Banco de Dados. Se a sentença for uma consulta, o resultado é apresentado. Se for um comando de atualização, ele é executado.

As consultas sempre são feitas em forma de regras, onde a cabeça da regra representa o conjunto de tuplas que se deseja receber como resposta. Como pode ser útil guardar os resultados de uma consulta em uma relação, de forma que eles possam ser utilizados no corpo de outra consulta, a interface interativa possui três comandos que permitem salvar resultados intermediários. Estes comandos são: *iniciar sessão*, que faz com que todas as consultas formuladas, desde o início da sessão até seu encerramento, sejam transformadas em relações derivadas, que serão virtuais e não atualizáveis; *encerrar sessão* que encerra a sessão, excluindo todas as relações derivadas criadas durante a sessão que não foram salvas; e *salvar sessão*, que torna as relações derivadas criadas durante a sessão, permanentemente parte do Banco de Dados Dedutivo.

### 8.1.3 Novos componentes Delphi

O sistema Delphi [BOR 95, BOR 95a, BOR 95b, RUB 95] utiliza o conceito de componentes no desenvolvimento das aplicações. Os componentes são classes com uma série de propriedade, métodos e eventos. As aplicações são construídas adicionando à aplicação os componentes necessários e utilizando suas propriedade, métodos e eventos. Exemplos de componentes são botões, janelas e menus.

No Delphi, existem, também, componentes criados especificamente para o desenvolvimento de aplicações de Banco de Dados. Entre estes, destacam-se o componente *TQuery*, utilizado para submeter comandos SQL ao Banco de Dados e o componente *TTable*, utilizado para manutenção de relações do Banco de Dados. No sistema DEDALO, foram criadas variações destes componentes que têm as mesmas finalidades, mas são capazes de reconhecer sentenças DEDALO. Os dois componentes criados são os seguintes:

? *TLogQuery*, que possui funcionamento idêntico ao componente *TQuery*, recebendo, no entanto, sentenças da linguagem DEDALO ao invés de sentenças SQL. O componente *TLogQuery* pode ser utilizado para recuperar dados de uma relação ou

para atualizar uma relação. Se uma operação de atualização é requisitada, todo controle necessário sobre as atualizações de um Banco de Dados Dedutivo (propagações, traduções, verificação de integridade, reparos) é realizado.

? *TLogTable*, que possui a mesma funcionalidade do componente *TTable*, realizando, no entanto, todo o controle necessário sobre as atualizações de um Banco de Dados Dedutivo. O componente *TLogTable* pode ser utilizado para recuperar informações e para atualizar relações.

Utilizando estes componentes, o usuário do sistema DEDALO pode construir suas aplicações utilizando o ambiente Delphi. Isto é uma grande vantagem, pois o Delphi é uma excelente ferramenta para a construção de aplicações cliente/servidor.

### 8.1.4 Tradutor de Sentenças DEDALO/SQL-ANSI

O tradutor de Sentenças DEDALO/SQL-ANSI, recebe uma sentença DEDALO normalizada e a traduz em uma sentença SQL-ANSI que será executada pelo Banco de Dados. O algoritmo de tradução, adaptado do algoritmo proposto em [BÖH 94, MAR 90], é o seguinte:

1. Uma regra de derivação normalizada na linguagem DEDALO é traduzida em um comando SELECT da forma:

*SELECT DISTINCT ...*
*FROM ...*
*WHERE ...*

Os argumentos da parte *SELECT* são os atributos das relações do corpo que estão ligados a variáveis da cabeça da regra. Se ocorrer uma função agregada ou uma operação aritmética na cabeça, adiciona-se a função ou a operação à lista *SELECT*, substituindo-se as variáveis envolvidas na função ou na operação pelo atributo do corpo que está ligado à variável. Se ocorrer uma constante na cabeça da regra, esta é adicionada à lista SELECT. O exemplo 8.3 demonstra a geração da parte *SELECT*.

**Exemplo 8.3:** A regra de derivação:

*PAMs(Nome:X,PAM:(Y+2*Z)/3) :- pacientes(Nome:X,PAS:Y,PAD:Z).*

seria traduzida em:

*SELECT DISTINCT a0.Nome, (a0.PAS + 2 * a0.PAD) / 3)*
*FROM pacientes a0*

As partes *FROM* e *WHERE* são construídas de acordo com as seguintes regras:

a) Primeiro, o nome de relação de cada literal positivo do corpo da regra é adicionado à lista *FROM* com um aliás. Se existe alguma variável na cabeça da regra que está ligada a um atributo que aparece apenas em literais negativos, acrescentar um

destes literais à lista *FROM*. Depois, os argumentos dos literais do corpo são processados. Se o argumento é:

i. uma constante, uma condição *WHERE* da forma *Nome-do-atributo= constante* é gerada.

ii. uma variável que não ocorreu antes, a variável é ligada ao respectivo nome de atributo.

iii. uma variável que ocorreu antes, é gerada uma condição *WHERE* da forma *Atributo1=Atributo2*, onde *Atributo1* é o atributo a que a variável foi ligada anteriormente e *Atributo2* é o atributo que está ligado à esta variável.

iiii. uma operação aritmética, é gerada uma condição *WHERE* da forma *Atributo=Operação*, onde *Atributo* é o atributo ligado à operação aritmética e *Operação* é a operação aritmética com as variáveis substituídas pelos atributos a que foram ligadas.

O exemplo 8.4 demonstra a criação das partes *FROM* e *WHERE* utilizando o passo (a) do algoritmo.

**Exemplo 8.4:** A seguinte regra de derivação recuperará os pacientes com pressão arterial média correta, que estão na UTI e seus respectivos médicos responsáveis:

*PacNaUtiComPamCorreta(Nome:N,Médico_responsável:M) :-*
*pacientes(Nome:M,PAS:S,PAD:D,PAM:(S+2*D)/3,Setor:'UTI',Médico:C),*
*médicos(Código:C,Nome:M).*

seria traduzida em:

*SELECT DISTINCT a0.Nome,a1.Nome*
*FROM pacientes a0, médicos a1*
*WHERE a0.PAM = (a0.PAS+2*a0.PAD)/3 AND a0.Setor = 'UTI' AND*
*a0.Médico=a1.Código*

b) Operações de comparação são acrescentadas às condições *WHERE*, substituindo suas variáveis pelos atributos correspondentes, como demonstra o exemplo 8.5.

**Exemplo 8.5:** A seguinte relação derivada:

*PacientesMenores(Nome:X) :- pacientes(Nome:X,Idade:Y) , Y < 18.*

seria traduzida em:

*SELECT DISTINCT a0.Nome*
*FROM pacientes a0*
*WHERE a0.Idade < 18*

c) Subexpressões negadas são traduzidas em condições *WHERE* da forma

*NOT EXISTS (*

*SELECT **
*FROM ...*
*WHERE ... )*

As partes *FROM* e *WHERE* da subconsulta SQL são determinadas pela tradução (recursiva) da subexpressão. (Note que as ligações de variáveis permanecem válidas durante a tradução da subexpressão). O exemplo 8.6 demonstra a tradução de um literal negado.

**Exemplo 8.6:** A seguinte relação derivada:

*MedicamentosControlados(Medicamento:X) :- Medicamentos(Nome:X) ,*
*not(liberados(Medicamento:X)).*

seria traduzida em:

*SELECT DISTINCT a0.Nome*
*FROM medicamentos a0*
*WHERE NOT EXISTS(SELECT **
*FROM liberados a1*
*WHERE a0.Nome=a1.Medicamento)*

d) Se ocorrerem anotações *@n* na cabeça da regra, acrescentar *ORDER BY posição-do-campo* , na ordem dos números *n*, como demonstra o exemplo 8.7.

**Exemplo 8.7:** A regra de derivação:

*IdadesPacientes(Nome:X@2,Idade:Y@1) :- pacientes(Nome:X,Idade:Y).*

seria traduzida em:

*SELECT DISTINCT a0.Nome,a0.Idade*
*FROM pacientes a0*
*ORDER BY 2,1*

e) Se ocorrem funções agregadas na cabeça da regra, uma cláusula *GROUP BY* deve ser acrescentada à sentença SQL. Os argumentos da cláusula *GROUP BY* são os atributos ligados às variáveis que aparecem na cabeça da regra, exceto as variáveis envolvidas nas funções. O exemplo 8.8 demonstra a tradução de uma regra de derivação com funções agregadas.

**Exemplo 8.8:** A seguinte regra de derivação:

*PacientesPorSetor(Setor:X,count(Y)) :- pacientes(Nome:Y,Setor:X).*

seria traduzida em:

*SELECT DISTINCT a0.setor,count(a0.Nome)*
*FROM pacientes a0*
*GROUP BY a0.setor*

2. Para cada literal positivo *q* no corpo da regra, repete-se a tradução, acrescentando-se *a.cf * CF* à lista *SELECT*, onde *a* é o aliás da relação *q*, *cf* é um atributo da relação *q* que contém o fator de certeza de cada tupla de *q*, e *CF* é o fator de certeza da regra. A seguir, acrescenta-se à cláusula *WHERE* uma condição *a1.cf <= a2.cf AND a1.cf <= a3.cf AND ... AND a1.cf <= an.cf*, para o primeiro predicado e *ai.cf < a1.cf AND ai.cf < a2.cf AND ... AND ai.cf < an.cf* para os demais. Cada uma das sentenças SQL geradas devem ser ligadas pelo operador *UNION*. Isto irá computar os fatores de certeza das relações derivadas. O fator de certeza escolhido é sempre o menor, pois a operação que une os literais é uma conjunção. O exemplo 8.9 demonstra uma tradução que calcula os fatores de certeza das tuplas derivadas.

**Exemplo 8.9:** A seguinte regra de derivação:

*DoseAdequada(Paciente:P,medicamento:M,Dose:D) :-*
*pacientes(Nome:P,Idade:I) , prescricoes(Paciente:P,medicamento:M),*
*dosesIndicadas(Idade:I,Medicamento:M,Dose:D) cf 0.8.*

seria traduzida em:

```
SELECT DISTINCT a0.Nome,a1.medicamento,a3.Dose,a0.cf * 0.8
FROM pacientes a0, prescricoes a1, dosesIndicadas a2
WHERE a0.Nome=a1.Paciente AND a0.Idade=a2.Idade AND
 a1.medicamento=a2.Medicamento AND a0.cf <= a1.cf AND a0.cf <= a2.cf
UNION
SELECT DISTINCT a0.Nome,a1.medicamento,a3.Dose,a1.cf * 0.8
FROM pacientes a0, prescricoes a1, dosesIndicadas a2
WHERE a0.Nome=a1.Paciente AND a0.Idade=a2.Idade AND
 a1.medicamento=a2.Medicamento AND a1.cf < a0.cf AND a1.cf < a2.cf
UNION
SELECT DISTINCT a0.Nome,a1.medicamento,a3.Dose,a2.cf * 0.8
FROM pacientes a0, prescricoes a1, dosesIndicadas a2
WHERE a0.Nome=a1.Paciente AND a0.Idade=a2.Idade AND
 a1.medicamento=a2.Medicamento AND a2.cf < a0.cf AND a2.cf < a1.cf
```

3. Para cada literal negativo acrescentar, além da tradução descrita no passo 2.c, onde o fator de certeza do literal negativo é zero, traduções que recuperam as tuplas do literal negativo que possuem fator de certeza menor que um. Estas traduções tratam o literal negativo como um literal positivo e na cláusula *WHERE*, a condição acrescentada para obter o menor fator de certeza deve considerar o fator do literal negativo como sendo *1 - a.cf*, onde *a* é o aliás do literal negativo. Além disso, no comando *SELECT* que seleciona o fator de certeza do literal negativo se este for o menor, acrescentar uma condição *1 - a.cf > 0*, para excluir da seleção as tuplas que possuem fator de certeza 1. O exemplo 8.10 demonstra a computação do fator de certeza de uma regra de derivação com um literal negativo.

**Exemplo 8.10:** A seguinte relação derivada:

*MedicamentosControlados(Medicamento:X) :- Medicamentos(Nome:X) ,*
*not(liberados(Medicamento:X)).*

seria traduzida em:

```
SELECT DISTINCT a0.Nome, a0.cf * 1
    FROM medicamentos a0
    WHERE NOT EXISTS(SELECT *
            FROM liberados a1 WHERE a0.Nome=a1.Medicamento)
UNION
SELECT DISTINCT a0.Nome, a0.cf * 1
FROM medicamentos a0, liberados a1
WHERE a0.Nome=a1.Medicamento AND a0.cf <= 1 - a1.cf
UNION
SELECT DISTINCT a0.Nome, a1.cf * 1
FROM medicamentos a0, liberados a1
WHERE a0.Nome=a1.Medicamento AND  1 - a1.cf < a0.cf
        AND  1 - a1.cf > 0
```

4) Os comandos *SELECT* gerados para cada regra de derivação que define a relação derivada, são concatenados em um único comando *SELECT* SQL usando o operador de união, como demonstra o exemplo 8.11.

**Exemplo 8.11:** As regras de derivação:

*PacientesNaUTI(Nome:X@1) :- pacientes(Nome:X, Setor:'UTI1').*
*PacientesNaUTI(Nome:X@1) :- pacientes(Nome:X,Setor:'UTI2').*

seriam traduzidas em:

```
SELECT DISTINCT a0.Nome, a0.cf * 1
FROM pacientes ao
WHERE a0.Setor = 'UTI1'
UNION
SELECT DISTINCT a0.Nome, a0.cf * 1
FROM pacientes ao   WHERE a0.Setor = 'UTI2'   ORDER BY 1
```

Como pode ocorrer de uma mesma tupla na relação derivada ter dois fatores de confiança diferentes, é necessário que se faça uma computação adicional selecionando o maior destes dois fatores. Isto é feito criando uma visão auxiliar que é definida pela tradução gerada e, a seguir, selecionando as tuplas com o maior fator. Esta situação é demonstrada no exemplo 8.12.

**Exemplo 8.12:** Para as seguintes regras de derivação:

*PacientesNaUTI(Nome:X) :- pacientes(Nome:X, Setor:'UTI1').*
*PacientesNaUTI(Nome:X) :- pacientes(Nome:X,Setor:'UTI2').*

seria criada a seguinte visão:

```
CREATE VIEW aux_PacientesNaUTI (Nome, cf) AS
SELECT DISTINCT a0.Nome, a0.cf * 1
FROM pacientes ao
WHERE a0.Setor = 'UTI1'
UNION
SELECT DISTINCT a0.Nome, a0.cf * 1
FROM pacientes ao
WHERE a0.Setor = 'UTI2'
```

e a seguinte seleção:

```
 SELECT DISTINCT a0.Nome,Max(a0.cf)
FROM aux_PacientesNaUTI a0
GROUP BY a0.Nome
```

Se uma relação derivada é virtual, suas regras de derivação são traduzidas e ela é armazenada em forma de visão. Se as regras do exemplo 8.12 definissem uma relação virtual, esta seria criada como:

```
CREATE VIEW PacientesNaUTI(Nome,cf) AS
 SELECT DISTINCT a0.Nome,Max(a0.cf)
FROM aux_PacientesNaUTI a0
GROUP BY a0.Nome
```

Se, por outro lado, a relação derivada for materializada, ela será armazenada como uma relação do Banco de Dados e, no momento da sua criação, as tuplas que farão parte dela serão computadas com um comando *INSERT* SQL. A partir daí, as técnicas de propagação se encarregarão de manter as tuplas da relação atualizadas. Se as regras do exemplo 8.12 definissem uma relação derivada materializada, esta seria criada como:

```
CREATE TABLE PacientesNaUTI(Nome character(30), cf float)
```

A seguir o seguinte comando iria inicializar a relação com as tuplas que podem ser derivadas do estado atual do Banco de Dados:

```
INSERT INTO PacientesNaUTI
        SELECT DISTINCT a0.Nome,Max(a0.cf)
        FROM aux_PacientesNaUTI a0
        GROUP BY a0.Nome
```

Depois disso, a visão *auxPacientesNaUti* poderia ser destruída.

As restrições de integridade, que são vistas como relações derivadas materializadas, são armazenadas como relações. Assim, para cada restrição de integridade, uma relação, que será vazia, é criada.

O algoritmo que traduz as operações de atualização da linguagem DEDALO é o seguinte:

1. Se a operação for uma inserção, definida pela sintaxe I.1, com a forma:

*ins* $p(atr_1{:}c_1,atr_2{:}c_2,...,atr_n{:}c_n)$.

onde *p* é uma relação básica ou derivada em que se deseja inserir novas tuplas, $atr_1...atr_n$ são nomes de atributos da relação e $c_1...c_n$ são constantes.

A inserção será traduzida em um comando *INSERT* SQL sobre a relação *p*, onde os valores da cláusula *VALUES* serão as constantes $c_1...c_n$:

*INSERT INTO p* $(atr_1, atr_2,...,atr_n)$ *VALUES* $(c_1,c_2,...,c_n)$

O exemplo 8.13 demonstra a tradução de uma solicitação de inserção usando a sintaxe I.1.

**Exemplo 8.13:** A solicitação de inserção:

*ins pacientes(nome:'joão', idade:25, setor:'UTI').*

seria traduzida em:

*INSERT INTO pacientes (nome,idade,setor,cf) VALUES ('joão',25,'UTI',1)*

Antes que a inserção seja realizada, é necessário verificar se já não existe na relação a mesma tupla com um fator de certeza diferente. Se existir, a tupla não poderá ser inserida.

2. Se a operação for uma inserção, definida pela sintaxe I.2, com a forma:

*ins* $p(atr_1{:}argc_1,...,atr_n{:}argc_n)$:-
$q_1(atr_1{:}arg_1,...,atr_n{:}arg_n)$ ,...,$q_n(atr_1{:}arg_1,...,atr_n{:}arg_n)$, *C, cf v.*

onde *p* é uma relação básica ou derivada em que se deseja inserir novas tuplas, $q_1...q_n$ é o corpo da regra de inserção, sendo que cada um dos $q_i$*s* representa uma relação básica ou derivada, $atr_1...atr_n$ são nomes de atributos da relação, $argc_1...argc_n$ são variáveis ligadas a atributos do corpo, ou constantes, ou operações aritméticas, ou funções agregadas, $arg_1...arg_n$ são variáveis ligadas a atributos, ou constantes, ou operações aritméticas, *C* é uma operação de comparação opcional, *cf* indica que a regra possui um fator de certeza, representado por *v* que é um número cujo valor é maior que 0 e menor ou igual a 1.

O corpo da regra de inserção será traduzido em um comando *SELECT* SQL que irá computar todas as tuplas que devem ser inseridas. A seguir, para cada tupla computada, é gerado um comando *INSERT* como descrito no passo 1 do algoritmo. Isto é necessário para que se possa verificar, antes de inserir a tupla, se já não existe a mesma tupla com um fator de certeza diferente na relação.

3. Se a atualização for uma alteração, definida pela sintaxe U.1, com a forma:

*upd* $p(atr_1{:}c_1,atr_2{:}c_2,...,atr_n{:}c_n)$.

onde *p* é uma relação básica ou derivada em que se deseja modificar tuplas, $atr_1...atr_n$ são nomes de atributos da relação e $c_1...c_n$ são constantes.

A alteração seria traduzida em um comando *UPDATE* SQL da seguinte forma:

*UPDATE p SET* $atr_1=c_1$ , $atr_2 = c_2$, ..., $atr_n=c_n$

O exemplo 8.14 demonstra a tradução de uma solicitação de atualização usando a sintaxe U.1.

**Exemplo 8.14:** A solicitação de atualização:

*upd pacientes(idade:20).*

seria traduzida em:

*UPDATE pacientes SET idade=20*

4. Se a atualização for uma alteração, definida pela sintaxe U.2, com a forma:

*upd* $p(atr_1{:}argc_1,...,atr_n{:}argc_n)$:-
$q_1(atr_1{:}arg_1,...,atr_n{:}arg_n)$ ,...,$q_n(atr_1{:}arg_1,...,atr_n{:}arg_n)$, *C.*

onde *p* é uma relação básica ou derivada em que se deseja modificar tuplas, $q_1...q_n$ é o corpo da regra de modificação, sendo que cada um dos $q_i$*s* representa uma relação básica ou derivada, $atr_1...atr_n$ são nomes de atributos da relação, $argc_1...argc_n$ são variáveis ligadas a atributos do corpo, ou constantes, ou operações aritméticas, ou funções agregadas, $arg_1...arg_n$ são variáveis ligadas a atributos, ou constantes, ou operações aritméticas e *C* é uma operação de comparação opcional.

O corpo da regra de atualização será traduzido em um comando *SELECT* SQL que irá computar todas as tuplas que devem ser alteradas. Os atributos selecionados pelo comando *SELECT* conterão a chave da relação que será atualizada. A seguir, para cada tupla computada, é gerado um comando *UPDATE*.

O exemplo 8.15 demonstra a tradução de uma solicitação de atualização usando a sintaxe U.2.

**Exemplo 8.15:** Para solicitação de atualização, onde *Nome* é a chave da relação *pacientes*:

*upd pacientes(conta: x + x * 0.10) :- pacientes(setor:z,conta:x) ,*
*setores(setor:z,MedicoResponsável:'Maria').*

seria gerado o seguinte comando *SELECT* SQL:

*SELECT DISTINCT a0.Nome*
*FROM pacientes a0, setores a1*
*WHERE a0.setor=a1.setor AND a1.MedicoResponsável='Maria'*

Supondo que aplicação deste comando sobre o Banco de Dados resultasse nas seguinte tuplas:

*pacientes('João').*
*pacientes('Pedro').*

Os seguintes comandos *UPDATE* seriam gerados:

*UPDATE pacientes SET conta = conta + conta * 0.10 WHERE Nome='João'*

*UPDATE pacientes SET conta = conta + conta * 0.10 WHERE Nome='Maria'*

5. Se a atualização for uma exclusão, definida pela sintaxe D.1, com a forma:

*del p.*

onde *p* é uma relação básica ou derivada de que se deseja excluir tuplas.

A exclusão seria traduzida em um comando *DELETE* SQL com a forma:

*DELETE FROM p*

6. Se a atualização for uma exclusão, definida pela sintaxe D.2, com a forma:

$del\ p(atr_1{:}c_1,atr_2{:}c_2,...,atr_n{:}c_n).$

onde *p* é uma relação básica ou derivada de que se deseja excluir tuplas, $atr_1...atr_n$ são nomes de atributos da relação e $c_1...c_n$ são constantes.

A exclusão seria traduzida em um comando *DELETE* SQL da seguinte forma:

$DELETE\ FROM\ p\ WHERE\ atr_1{=}c_1\ ,\ atr_2 = c_2, ..., atr_n{=}c_n$

O exemplo 8.16 demonstra a tradução de uma solicitação de exclusão usando a sintaxe D.2.

**Exemplo 8.16:** A solicitação de exclusão:

*del pacientes(setor:'UTI').*

seria traduzida em:

*DELETE FROM pacientes WHERE setor = 'UTI'*

7. Se a atualização for uma exclusão, definida pela sintaxe D.3, com a forma:

$del\ p(atr_1{:}argc_1,...,atr_n{:}argc_n)\ :-$

$q_1(atr_1{:}arg_1,...,atr_n{:}arg_n),...,q_n(atr_1{:}arg_1,...,atr_n{:}arg_n),\ C.$

onde *p* é uma relação básica ou derivada de que se deseja excluir tuplas, $q_1...q_n$ é o corpo da regra de exclusão, sendo que cada um dos $q_i s$ representa uma relação básica ou derivada, $atr_1...atr_n$ são nomes de atributos da relação, $argc_1...argc_n$ são variáveis ligadas a atributos do corpo, ou constantes, ou operações aritméticas, ou funções agregadas, $arg_1...arg_n$ são variáveis ligadas a atributos, ou constantes, ou operações aritméticas e *C* é uma operação de comparação opcional.

O corpo da regra de exclusão será traduzido em um comando *SELECT* SQL que irá computar todas as tuplas que devem ser excluídas. Os atributos selecionados pelo comando *SELECT* conterão a chave da relação em que serão feitas as exclusões. A seguir, para cada tupla computada, é gerado um comando *DELETE*.

O exemplo 8.17 demonstra a tradução de uma solicitação de atualização usando a sintaxe D.3.

**Exemplo 8.17:** Para solicitação de exclusão, onde *Nome* é a chave da relação *pacientes*:

*del pacientes(Nome:X) :- pacientes(Nome:X,Idade:Y), Y < 15.*

seria gerado o seguinte comando *SELECT* SQL:

*SELECT DISTINCT a0.Nome*
*FROM pacientes a0*
*WHERE a0.idade < 15*

Supondo que aplicação deste comando sobre o Banco de Dados resultasse nas seguinte tuplas:

*pacientes('Maria').*
*pacientes('Carlos').*

Os seguintes comandos *DELETE* seriam gerados:

*DELETE FROM pacientes WHERE Nome='Maria'*

*DELETE FROM pacientes WHERE Nome='Carlos'*

## 8.2 A implementação do Sistema DEDALO

O objetivo desta seção é comentar alguns detalhes de implementação ainda não discutidos anteriormente. Estes detalhes se dividem entre dois tópicos: o processamento das consultas submetidas ao sistema e a forma de processamento das atualizações solicitadas. Cada um destes tópicos será apresentado a seguir.

### 8.2.1 Processamento de consultas

Quando uma consulta é submetida ao sistema DEDALO, é criado o grafo de dependências para esta consulta. A partir deste grafo, é possível conhecer todas as relações de que a regra que representa a consulta depende. Se todas estas relações forem básicas, não recursivas ou materializadas, o comando SQL correspondente à consulta é simplesmente executado. Se uma das relações de que depende a consulta é recursiva e virtual, porém, deve-se computá-la antes de poder responder à consulta formulada.

A computação de uma relação derivada recursiva é feita da seguinte forma:

1. Uma relação auxiliar, cujo esquema é o esquema da relação recursiva, é criada.

2. Uma visão auxiliar, que é definida pelos comandos SQL que definem a relação derivada recursiva é criada.

3. Uma série de comandos *INSERT*, que inserem as tuplas da visão auxiliar na relação auxiliar são executados, até que nenhuma nova tupla possa ser inserida na relação.

4. Por fim, uma visão definida sobre a relação auxiliar, que recupera somente as tuplas com os maiores fatores de certeza, é criada. Esta visão será a relação derivada recursiva.

No momento da criação da relação derivada recursiva, a visão auxiliar, a relação auxiliar e a visão que representará a relação recursiva são criadas. Porém, como a relação deve ser virtual, a relação auxiliar permanece vazia até que uma consulta seja solicitada. O exemplo 8.18 demonstra a computação de uma relação derivada recursiva virtual.

**Exemplo 8.18:** Considerando as seguintes regras de derivação que definem uma relação derivada recursiva virtual:

*ancestrais(Ancestral:X,Descendente:Y) :- pacientes(Nome:Y,Pai:X).*

*ancestrais(Ancestral:X,Descendente:Y) :-*
*ancestrais(Ancestral:X,Descendente:Z) ,*
*ancestrais(Ancestral:Z,Descendente:Y).*

A seguinte relação seria criada:

```
CREATE TABLE RAux_ancestrais (Ancestral character(30),
        Descendente   character(30), cf float)
```

A seguinte visão auxiliar seria criada:

```
CREATE VIEW Aux_ancestrais (Ancestral, Descendente, cf) AS
    SELECT DISTINCT a0.Pai, ao.Nome, ao.cf * 1
    FROM pacientes a0
    UNION
    SELECT DISTINCT a0.Ancestral, a1.Descendente, a0.cf * 1
    FROM RAux_ancestrais a0, Raux_ancestrais a1
    WHERE a0.Descendente = a1.Ancestral AND a0.cf <= a1.cf
    UNION
     SELECT DISTINCT a0.Ancestral, a1.Descendente, a1.cf * 1
    FROM RAux_ancestrais a0, Raux_ancestrais a1
    WHERE a0.Descendente = a1.Ancestral AND a1.cf < a0.cf
```

Supondo que o conteúdo da relação *pacientes* é:

*pacientes('João','Pedro',1)*
*pacientes('Pedro','Carlos',1)*

A visão *Aux_Ancestrais* conteria:

*Aux_Ancestrais('Pedro','João',1)*
*Aux_Ancestrais('Carlos','Pedro',1)*

A seguir, o seguinte comando seria executado:

*INSERT INTO RAux_Ancestrais SELECT * FROM Aux_Ancestrais*

A relação *RAux_Ancestrais* passaria a conter as tuplas:

*RAux_Ancestrais('Pedro','João',1)*
*RAux_Ancestrais('Carlos','Pedro',1)*

Conseqüentemente, a visão *Aux_Ancestrais* passaria a conter as tuplas:

*Aux_Ancestrais('Pedro','João',1)*
*Aux_Ancestrais('Carlos','Pedro',1)*
*Aux_Ancestrais('Carlos','João',1)*

Uma nova execução do comando:

*INSERT INTO RAux_Ancestrais SELECT * FROM Aux_Ancestrais*

faria com que o conteúdo da relação *RAux_Ancestrais* ficasse sendo:

*RAux_Ancestrais('Pedro','João',1)*
*RAux_Ancestrais('Carlos','Pedro',1)*
*RAux_Ancestrais('Carlos','João',1)*

Como uma nova execução do comando *INSERT* não acrescentaria novas tuplas à relação *RAux_Ancestrais*, o processamento encerra. A seguir, poderá ser criada a visão que representará a relação derivada recursiva:

```
CREATE VIEW Ancestrais(Ancestral,Descendente,cf) AS
        SELECT DISTINCT a0.Ancestral, a0.Descendente,Max(a0.cf)
        FROM RAux_Ancestrais a0
        GROUP BY a0.Ancestral, a0.Descendente
```

Quando a consulta for respondida, as tuplas da relação auxiliar são excluídas. Em função do processamento da relação derivada recursiva poder se tornar bastante lento, é importante considerar a possibilidade de definir as relações derivadas recursivas como materializadas.

### 8.2.2 Processamento de atualizações

Como foi citado nos capítulos anteriores, o sistema DEDALO utiliza relações delta no gerenciamento das atualizações sobre as relações do Banco de Dados. As relações delta são relações que armazenam as tuplas inseridas e excluídas durante uma transação. Assim, para cada relação, básica ou derivada, existem duas relações delta. Estas relações permanecem vazias até que uma transação inicie. Durante a transação elas recebem tuplas e, ao final, são novamente esvaziadas.

Durante uma transação, cada tupla que é inserida, é inserida também na relação delta de inserção. Da mesma forma, cada tupla que é excluída, é inserida na relação delta de exclusão. As alterações, no entanto, apesar se serem traduzidas em comandos *UPDATE* SQL, são tratadas pelo sistema como exclusões seguidas de

inserções. Então, para cada tupla alterada, a tupla antiga é inserida na relação delta de exclusão e a tupla alterada é inserida na relação delta de inserção.

Os novos componentes Delphi criados realizam todo o controle necessário sobre as atualizações. A integridade do Banco de Dados Dedutivo só poderá ser garantida se todas as atualizações realizadas sobre o BD forem feitas através das ferramentas fornecidas pelo Sistema DEDALO, pois estas estão preparadas para realizar todo o processo necessário para manter as relações derivadas corretas e as restrições de integridade não violadas.

## 8.3 Comentários finais

A arquitetura do Sistema DEDALO permite que ele possa ser utilizado sobre um grande número de Sistemas Gerenciadores de Bancos de Dados, transformando estes sistemas em Sistemas de Bancos de Dados Dedutivos.

O tratamento do raciocínio aproximado torna o tempo de respostas às consultas sensivelmente mais lento, em função da necessidade de criação de visões auxiliares e da filtragem dos maiores fatores de certeza. Em função na natureza das aplicações que se pretende atender com Sistema DEDALO, que são aplicações com características de típicas de sistemas especialistas, optou-se por adicionar esta funcionalidade, em detrimento da performance. Nesta implementação do sistema, mesmo as aplicações que não fazem uso de raciocínio aproximado sofrerão seus efeitos. Em trabalhos futuros, pretende-se criar uma nova versão do Sistema DEDALO sem tratamento de raciocínio aproximado, que beneficiará as aplicações que não necessitam desta característica.

As regras da aplicação utilizada para testar a aplicabilidade do Sistema DEDALO, descrita no capítulo 9, não fazem uso de raciocínio aproximado, o que limitou uma maior exploração desta facilidade. No entanto, foram submetidas ao sistema algumas regras que utilizam raciocínio aproximado que comprovaram a adequabilidade da solução proposta. No exemplo 8.19 são apresentadas algumas destas regras.

**Exemplo 8.19:** O seguinte conjunto de regras verifica se o deslocamento de uma cidade a outra pode ser feito em uma viagem rápida. Se houver um vôo direto entre as duas cidades, a viagem é considerada rápida com certeza absoluta, se não, o fator de certeza diminui de acordo com o número de vôos indiretos que são necessários para atingir o destino.

*ViagemRápida(Origem:X,Destino:Y) :-*
*VôoDireto(Origem:X,Destino:Y) cf 1.*

*ViagemRápida(Origem:X,Destino:Y) :-*
*VôoDireto(Origem:X,Destino:Z),*
*ViagemRápida(Origem:Z,Destino:Y) cf 0.9.*

Supondo que a relação *VôoDireto* contenha as seguintes tuplas:

*VôoDireto(‘Porto Alegre’,’São Paulo’).*
*VôoDireto(‘São Paulo’,’Los Angeles’).*
*VôoDireto(‘Porto Alegre’,’Rio de Janeiro’).*
*VôoDireto(‘Rio de Janeiro’,’São Paulo’).*

A relação *ViagemRápida* conteria as tuplas:

*ViagemRápida(‘Porto Alegre’,’São Paulo’,1).*
*ViagemRápida(‘São Paulo’,’Los Angeles’,1).*
*ViagemRápida(‘Porto Alegre’,’Rio de Janeiro’,1).*
*ViagemRápida(‘Rio de Janeiro’,’São Paulo’,1).*
*ViagemRápida(‘Porto Alegre’,’Los Angeles’,0.9).*
*ViagemRápida(‘Porto Alegre’,’São Paulo’,0.9).*
*ViagemRápida(‘Rio de Janeiro’,’Los Angeles’,0.9).*
*ViagemRápida(‘Porto Alegre’,’Los Angeles’,0,81).*

Como apenas as tuplas com o maior fator de certeza são mantidas na relação, a relação *ViagemRápida* final seria:

*ViagemRápida(‘Porto Alegre’,’São Paulo’,1).*
*ViagemRápida(‘São Paulo’,’Los Angeles’,1).*
*ViagemRápida(‘Porto Alegre’,’Rio de Janeiro’,1).*
*ViagemRápida(‘Rio de Janeiro’,’São Paulo’,1).*
*ViagemRápida(‘Porto Alegre’,’Los Angeles’,0.9).*
*ViagemRápida(‘Rio de Janeiro’,’Los Angeles’,0.9).*

O Sistema DEDALO não realiza nenhum tipo de otimização de consultas. As consultas são traduzidas em comandos SQL que serão otimizados pelo Sistema Gerenciador do Banco de Dados. No entanto, existem algumas otimizações que poderiam ser feitas pelo Sistema DEDALO, principalmente no processamento das consultas recursivas. Estas otimizações deverão ser propostas em futuros trabalhos, que buscarão aumentar a eficiência do processamento de consultas no Sistema DEDALO.

# 9 Aleph - Uma aplicação utilizando o Sistema DEDALO

As aplicações de Bancos de Dados Dedutivos são, em princípio, todas as aplicações de Bancos de Dados tradicionais, que se beneficiariam com a possibilidade de formular consultas recursivas, definir regras de restrições de integridade e utilizar uma linguagem baseada em lógica declarativa. Outras aplicações, que exigem funcionalidades não fornecidas por Sistemas de Bancos de Dados tradicionais, poderiam encontrar em um Sistema de Banco de Dados Dedutivo uma ferramenta adequada para seus propósitos. As aplicações tradicionalmente citadas como aplicações típicas de SBDD são [TSU 91]: bancos de dados científicos, análise exploratória de dados, controle de processos e aplicações que fazem uso constante de consultas recursivas, como o problema da lista de materiais.

Aplicações que possuem características de sistemas especialistas, como necessidade de dedução e raciocínio aproximado, mas que, por tratarem de um grande volume de dados, necessitam também da funcionalidade de um Sistema Gerenciador de Banco de Dados, são outra classe de aplicações que seriam bem atendidas por um Sistema de Banco de Dados Dedutivo.

O objetivo deste capítulo é apresentar uma aplicação que possui características de aplicações típicas de Bancos de Dados convencionais, pois trata de um grande volume de dados, em constante modificação, e características de sistemas especialistas, pois realiza inferências sobre os dados, deduzindo novas informações. A aplicação escolhida para demonstrar a funcionalidade do Sistema DEDALO nestas condições, é o Sistema Aleph [NEV 95]. O Sistema Aleph (Alertas Pharmacêuticos) é um sistema de alerta na prescrição e dispensação de medicamentos cardiovasculares, desenvolvido como um trabalho de dissertação de mestrado no curso de Pós-Graduação em Ciências Farmacêuticas da Universidade Federal do Rio Grande do Sul. O sistema Aleph foi totalmente reimplementado utilizando o Sistema DEDALO, o que permitiu uma comparação entre a implementação de uma aplicação deste tipo utilizando um Sistema de Banco de Dados convencional e um Sistema de Banco de Dados Dedutivo.

Na seção 9.1 será feita uma breve apresentação do sistema Aleph. Na seção 9.2 será apresentado o mecanismo de raciocínio utilizado pelo Aleph. Na seção 9.3, a utilização das regras DEDALO como mecanismo de raciocínio no sistema Aleph será

discutida. Por fim, na seção 9.4, será apresentada a implementação do sistema Aleph com o Sistema DEDALO.

## 9.1 O Sistema Aleph

O objetivo do Sistema Aleph é fornecer ao usuário acesso a uma base de dados que contenha informações sobre medicamentos cardiovasculares e alerte o usuário a cerca de interações entre medicamentos, posologias inadequadas, contra indicações, entre outros.

O sistema Aleph possui um módulo de consultas, que fornece informações clínicas sobre fármacos, indicações clínicas de medicamentos e contra indicações. Além das consultas, o sistema Aleph possui um cadastro de pacientes e prescrições. O cadastro de pacientes armazena informações sobre o paciente e o cadastro de prescrições permite que se prescrevam medicamentos a ele. Para cada prescrição, o sistema é capaz de realizar uma análise sobre a posologia, interações, efeitos adversos, contra indicações e educação e cuidado do paciente.

As consultas e a manutenção das informações do paciente e das prescrições caracterizam aplicações tipicamente resolvidas por Sistemas de Bancos de Dados e a inferência que é realizada sobre a prescrição, analisando-a e gerando alertas, é uma aplicação tipicamente resolvida por sistemas especialistas.

O sistema Aleph foi desenvolvido em Visual Basic, utilizando o Banco de Dados MS-Access.

## 9.2 O raciocínio no sistema Aleph

Quando uma análise de posologia é solicitada, o sistema Aleph dispara um conjunto de regras que, em função das informações do paciente e das doses dos medicamentos prescritas, gera uma série de alertas que informarão o usuário a respeito de doses inadequadas, presença de interações com outros medicamentos, inadequação da forma prescrita do medicamento, entre outros. As regras que geram estes alertas foram elaboradas sob a forma de *MLMs* (Medical Logic Modules) [AGU 91, HRI 90].

MLMs são módulos utilizados para representar conhecimento em sistemas de informação na área médica. Cada módulo contém conhecimento suficiente para chegar a uma pequena conclusão. O uso destes módulos permite o compartilhamento de conhecimento na área médica entre instituições. Exemplos de conhecimento que podem ser representados são sugestões terapêuticas, alertas de contra indicações e interpretação de dados laboratoriais.

Um MLM é composto de uma série de *slots* que armazenam informações sobre o MLM. Estes *slots* são agrupados em três categorias:

? *maintenance*, que contém slots que especificam informações de manutenção que não está relacionada com o conhecimento médico contido no módulo. Os slots desta categoria são: *title*, que guarda o título do módulo; *filename*, que contém o nome do arquivo em que o módulo está armazenado; *version*, que contém a versão do módulo; *institution*, que armazena a instituição que criou o MLM, *author*, que contém o nome do autor do módulo; *specialist*, que contém o nome da pessoa responsável pela validação do módulo na instituição; *date*, que contém a data de criação do módulo e *validation*, que determina o nível de validação do módulo, que pode ser *production*, *research*, *testing* ou *expired*.

? *Library*, que contém slots pertinentes à manutenção do conhecimento contido no módulo. Estes slots são: *purpose*, que é um comentário sobre o objetivo do módulo; *keywords*, que contém palavras chave sobre o assunto que trata o módulo; *citations*, que contém referências bibliográficas sobre o conhecimento do módulo; e *links*, que contém ligações com outras fontes, tais como livros eletrônicos.

? *Knowledge*, que contém o conhecimento médico representado no MLM. Os slots desta categoria são: *type*, que identifica a forma que o MLM será utilizado; *data*, que define os termos utilizados no restante do módulo, ou seja, são variáveis locais que recebem resultados de consultas sobre a base de dados; *evoke*, que contém a condição que fará o MLM ser ativado; *logic*, que contém a regra médica ou condição médica que será testada; e *action*, que especifica o que deve ser feito se a premissa for satisfeita.

No sistema Aleph, os MLMs foram implementados como sub rotinas da linguagem Visual Basic, onde os slots das categorias *maintenance* e *library* e o slot *type* da categoria *knowledge* são comentários. O slot *data* são variáveis que recebem valores obtidos de consultas sobre o Banco de Dados. O slot *evoke* é apenas um comentário, pois a ativação da sub rotina é feita na implementação do sistema. O slot *logic* é uma série de sentenças *IF...THEN...ELSE*, que testam os valores das variáveis e geram um alerta, que é uma mensagem descrevendo a conclusão atingida. O slot *action* é a chamada de uma outra sub rotina que armazena o alerta em uma relação especial. O exemplo 9.1 apresenta um dos MLMs do sistema Aleph.

**Exemplo 9.1:** A seguinte subrotina analisa a prescrição de comprimidos de digoxina para crianças entre 2 e 5 anos ( ' indica comentário ):

```
Sub DigoxinaComp025_2a5 ()
'Maintenance:
        'title: Prescrição de digoxina comprimidos para crianças entre 2 e 5
anos
        'filename: digoxina.bas
        'version: 1
        'institution: UFRGS
        'author: Eugênio Rodrigo Zimmer Neves
        'specialist:
        'date: 1995
        'validation: testing;
```

```
'library:
        'purpose: analisar prescrição de comprimidos de digoxina para crianças entre 2 e 5 anos
        'keywords: digoxina
        'citations:
        'links:

'knowledge:
    'type: data-driven
    'data:
        DoseUnitaria = 250   'mcg
        DoseDiaria=NumeroDeUnidadesPorTomada*VezesAoDia* DoseUnitaria
        DosePorQuilo = DoseDiaria / PesoPac
    'evoke: prescrição de digoxina comprimidos para crianças entre 2 e 5 anos
    'logic:
        If DosePorQuilo > 40 Then
                Msg = "   De acordo com os dados da base de conhecimentos esta parece ser uma dose excessiva para uma criança desta idade."
        ElseIf DosePorQuilo < 30 Then
                Msg = "   De acordo com os dados da base de conhecimentos esta parece ser uma dose insuficiente para uma criança desta idade."
        Else
                Msg = " De acordo com os dados disponíveis na base de conhecimento esta parece ser uma dose adequada de digoxina para uma criança desta idade e deste peso (dose de digoxina entre 30 e 40 µg por quilo de peso corporal)."
        End If

    'action:
        ColocaAlerta Msg
End Sub
```

Este MLM analisa a dose de digoxina prescrita para crianças entre 2 e 5 anos e gera uma mensagem de alerta em função da dose prescrita. As variáveis *NumeroDeUnidadesPorTomada*, *VezesAoDia* e *PesoPac*, correspondem a atributos das relações *prescricoes* e *pacientes*. A ação é a chamada para uma sub rotina que armazena a mensagem em uma relação chamada *Alertas*. Esta relação permanece sempre vazia e é preenchida quando uma análise é solicitada.

## 9.3 A utilização de regras DEDALO no sistema Aleph

As regras DEDALO foram utilizadas no sistema Aleph em 3 situações:

? Para a criação das visões presentes na base de dados;

? Para a criação de restrições de integridade, que eram verificadas via programação;

? Para a implementação do raciocínio contido nos MLMs.

Na primeira situação, para adaptar os conceitos de um Sistema de Banco de Dados tradicional para os conceitos de um Sistema de Banco de Dados Dedutivo, as visões que faziam parte do BD foram transformadas em relações derivadas virtuais do Sistema DEDALO. Alguns exemplos desta transformação são:

*CREATE VIEW EfeitosAdversos AS SELECT DISTINCT EfAdv.Cont,*
*EfAdv.EfeitoAdverso, EfAdvSint.Sintoma FROM EfAdv, EfAdvSint*
*WHERE EfAdv.Cont = EfAdvSint.Cont*
*ORDER BY EfAdv.EfeitoAdverso, EfAdvSint.Sintoma*

que foi transformada em:

*EfeitosAdversos(Cont:C,EfeitoAdverso:E@1,Sintoma:S@2) :-*
*EfAdv(Cont:C,EfeitoAdverso:E) , EfAdvSint(Cont:C,Sintoma:S).*

*CREATE VIEW NomeDosGenericos AS*
*SELECT DISTINCT Genericos.Complemento, Genericos.Descritor,*
*Genericos.GenericoID,Genericos.NomeGenerico,*
*Genericos.UnidadeFormaFarmaceutica,*
*FormasFarmaceuticas.UnidadeAplicacao,*
*CompoGenericos.ValorConc, CompoGenericos.UniConc*
*FROM Genericos, CompoGenericos, FormasFarmaceuticas*
*WHERE Genericos.GenericoID = CompoGenericos.GenericoID AND*
*CompoGenericos.CodFormaFarmaceutica=*
*FormasFarmaceuticas.CodFormaFarmaceutica*

que foi transformada em:

*NomeDosGenericos(Complemento:C,Descritor:D,GenericoID:G,*
*NomeGenerico:N, UnidadeFormaFarmaceutica:U,*
*UnidadeAplicacao:UA,ValorConc:V,UniConc:UC) :-*

*genericos(Complemento:C,Descritor:D,GenericoID:G,NomeGenerico:N,*
*UnidadeFormaFarmaceutica:U) ,*
*CompoGenericos(ValorConc:V,UniConc:UC,*
*CodFormaFarmaceutica:CF,GenericoID:G) ,*

*FormasFarmaceuticas(UnidadeAplicacao:UA,*
*CodFormaFarmaceutica:CF).*

As restrições de integridade, que antes eram mantidas via programação, foram transformadas em regras de restrições de integridade. Algumas das regras de integridade que foram criadas são:

*HomemGravido(PacID:P) :-*
*pacientes(PacID:P,Homem:'Sim',Gravidez:'Sim').*

*PacienteSemPeso(PacID:P) :- pacientes(PacID:P,PesoPac:0).*

Cada MLM do sistema Aleph foi transformado em regras de derivação que definem a relação derivada *Alertas*. Esta relação recebe os alertas gerados a partir das análises realizadas sobre as informações contidas no Banco de Dados. O MLM do

exemplo 9.1 foi transformado nas seguintes regras de derivação:

*Alertas(Paciente:PacID,MLM:1,Mensagem:'De acordo com os dados da base de conhecimentos esta parece ser uma dose excessiva para uma criança desta idade.') :- pacientes(PacID:P,PesoPac:Peso) ,*
*prescricoes(NumeroDeUnidadesPorTomada:N,VezesAoDia:V) ,*
*(N * V * 250) / PesoPac > 40.*

*Alertas(Paciente:PacID,MLM:1,Mensagem:'De acordo com os dados da base de conhecimentos esta parece ser uma dose insuficiente para uma criança desta idade.') :- pacientes(PacID:P,PesoPac:Peso) ,*
*prescricoes(NumeroDeUnidadesPorTomada:N,VezesAoDia:V) ,*
*(N * V * 250) / PesoPac < 30.*

*Alertas(Paciente:PacID,MLM:1,Mensagem:'De acordo com os dados disponíveis na base de conhecimento esta parece ser uma dose adequada de digoxina para uma criança desta idade e deste peso (dose de digoxina (entre 30 e 40 µg por quilo de peso corporal).') :-*
*pacientes(PacID:P,PesoPac:Peso) ,*
*prescricoes(NumeroDeUnidadesPorTomada:N,VezesAoDia:V) ,*
*(N * V * 250) / PesoPac >= 30 , (N * V * 250) / PesoPac <= 40.*

São duas as principais vantagens da utilização de regras de derivação na geração dos alertas:

? As regras que geram os alertas deixam de fazer parte do programa de aplicação e passam a fazer parte do esquema do BDD, o que facilita sua manutenção;

? A relação *Alertas* estará sempre correta e atualizada quando o Banco de Dados for modificado.

Como as ferramentas utilizadas no desenvolvimento do Sistema Aleph não possuíam suporte para raciocínio aproximado, este não foi implementado, apesar de ser reconhecida a sua necessidade. Com a possibilidade de utilização deste tipo de raciocínio no Sistema DEDALO, uma nova aquisição de conhecimento deve ser feita para que se possa incorporar esta característica ao sistema Aleph. Apesar desta característica não ter sido utilizada na implementação da aplicação, foram submetidas ao Sistema DEDALO regras que faziam uso de raciocínio aproximado para que esta característica fosse testada. Apesar dos testes realizados terem apresentado resultados satisfatórios, ainda é necessário que regras de uma aplicação real utilizando raciocínio aproximado sejam submetidas ao sistema para que a adequabilidade desta solução possa ser melhor comprovada.

## 9.4 A construção e o funcionamento do sistema Aleph com o DEDALO

Para implementar o sistema Aleph utilizando o Sistema Gerenciador de Bancos de Dados Dedutivos DEDALO, foi necessário, em primeiro lugar, transportar

os dados do Banco de Dados MS-Access para um outro Banco de Dados, pois o MS-Access não é compatível com o padrão SQL-ANSI. O SGBD escolhido para armazenar os dados do Sistema Aleph foi o Sistema Gerenciador de Banco de Dados Relacional Watcom SQL.

Depois de transportar as relações básicas para o novo Banco de Dados, foram criadas as relações derivadas correspondentes às antigas visões, utilizando a linguagem DEDALO. A seguir, foram definidas as restrições de integridade, que eram mantidas via programação, através de regras de integridade. Finalmente, as regras correspondentes aos MLMs foram criadas. Com o Banco de Dados definido, o sistema Aleph foi totalmente reimplementado utilizando o ambiente Delphi e os componentes *TLogQuery* e *TLogTable*.

Com a definição das restrições de integridade e alertas por meio de regras, a implementação se resumiu a consultar dados e gerar manutenções utilizando os componentes criados pelo sistema DEDALO. A cada modificação que é realizada sobre a base, as restrições de integridade são verificadas e a relação *Alertas* é atualizada.

Para que os mecanismos de propagação de atualizações para relações derivadas materializadas pudessem ser testados, algumas das visões foram transformadas em relações derivadas materializadas. Com o propósito de testar os mecanismos de tradução das atualizações sobre relações derivadas, algumas destas relações foram selecionadas para serem atualizáveis. Algumas das restrições de integridade foram selecionadas para serem reparadas em caso de violação, para que os mecanismos de reparo pudessem ser testados.

## 9.5 Comentários finais

A implementação de uma aplicação real utilizando o Sistema DEDALO serviu para que as potencialidades do sistema fossem testadas e para que os mecanismos criados para manter a integridade das informações fossem postos a prova. Como resultado, verificou-se que a aplicação obteve benefícios com a utilização do Banco de Dados Dedutivo, principalmente sob dois aspectos: a possibilidade de definição de restrições de integridade através de regras e na utilização de regras de derivação para implementar os MLMs.

As vantagens de se poder definir restrições de integridade através de regras são óbvias: o programa de aplicação não precisa se encarregar em manter a integridade das restrições, e a criação, modificação ou eliminação de restrições não exigirá que se modifique os programas de aplicação.

A utilização de regras de derivação na implementação dos MLMs apresenta três vantagens:

? O programa de aplicação não precisa se preocupar com a ativação do módulo, pois a relação derivada que armazena os resultados do MLM, em função dos mecanismos descritos anteriormente, estará sempre correta ;

? A criação, modificação e eliminação de MLMs poderão ser feitas a qualquer momento, sem que haja necessidade de modificação dos programas de aplicação;

? O autor do MLM não precisa se preocupar com a recuperação das informações contidas no Banco de Dados, pois isto é feito implicitamente pela regra de derivação.

A implementação desta aplicação provou que os mecanismos de controle de atualizações e de restrições de integridade são capazes de manter corretamente a integridade das informações contidas no BDD, e que as extensões criadas na linguagem DEDALO são adequadas para atender aplicações do mundo real.

# 10 Conclusões e trabalhos futuros

Este trabalho apresenta um estudo sobre os Sistemas de Bancos de Dados Dedutivos, discutindo alguns dos problemas encontrados na área. Em particular, as deficiências da linguagem Datalog pura foram identificadas e uma nova linguagem, que estende o Datalog, foi proposta, introduzindo negação, operações aritméticas, operações de comparação, funções agregadas e raciocínio aproximado, além de criar uma notação que permite referenciar os atributos pelo nome e não por sua posição dentro da relação. As atualizações em Bancos de Dados Dedutivos também foram estudadas, sendo analisados e implementados métodos que realizam a propagação de atualizações para relações derivadas materializadas. Também foram propostos e implementados métodos que traduzem as atualizações sobre relações derivadas em termos de atualizações em relações básicas. A linguagem criada também permite a especificação de restrições de integridade, sendo que métodos para sua verificação e para a geração de reparos para transações que violam restrições foram propostos.

Neste trabalho, também é apresentado o Sistema Gerenciador de Banco de Dados Dedutivo DEDALO, que é formado por um conjunto de ferramentas que implementam as idéias descritas.

Em relação aos objetivos propostos no capítulo 1, pode-se concluir que:

1. A linguagem DEDALO estende o poder expressivo do Datalog, mostrando-se mais adequada para uso em situações práticas, em função da notação criada e das extensões propostas;

2. O problema das atualizações em Bancos de Dados Dedutivos foi estudado e as soluções analisadas, propostas e implementadas são capazes de resolvê-lo com confiabilidade e eficiência;

3. Foram criados mecanismos para especificação de restrições de integridade bem como métodos de verificação de integridade eficientes;

4. Foram propostos e implementados mecanismos de reparo para transações que violam as restrições de integridade definidas no sistema;

5. O Gerenciador de Regras, ferramenta que faz parte do sistema DEDALO, auxilia na criação e manutenção da base de regras e das restrições de integridade;

6. Um protótipo foi implementado, onde os estudos realizados foram postos em prática. A arquitetura do sistema permite que ele possa ser utilizado em conjunto

com qualquer Sistema Gerenciador de Banco de Dados relacional que possua interface com ODBC e seja compatível com o padrão SQL-ANSI.

Com este trabalho, foi demonstrado a aplicabilidade dos Sistemas de Bancos de Dados Dedutivos em situações práticas. A aplicação descrita no capítulo 9 foi beneficiada com a utilização do protótipo desenvolvido, demonstrando que muitas outras aplicações reais poderiam obter os mesmos benefícios.

## 10.1 Contribuições deste trabalho

As principais contribuições deste trabalho são:

1. A definição de uma nova linguagem baseada em lógica, com maior poder expressivo que o Datalog e mais adequada à utilização em aplicações reais;

2. O estudo, análise, implementação e validação dos métodos de propagação de atualizações para relações derivadas materializadas;

3. O estudo, análise, implementação e validação de métodos de geração de traduções para atualizações sobre relações derivadas;

4. O estudo, análise, implementação e validação de métodos de verificação de restrições de integridade e de técnicas de reparos para transações que levam o Banco de Dados a um estado inconsistente;

5. A implementação de um protótipo de Sistema de Banco de Dados Dedutivo que utiliza todos os estudos realizados e que possui características que beneficiam aplicações reais;

6. A implementação de uma aplicação real utilizando o protótipo construído, que prova a adequabilidade das soluções propostas.

## 10.2 Trabalhos futuros

Estudos sobre a otimização das consultas no Sistema DEDALO devem ser feitos para que este se torne mais eficiente. Otimização de consultas em Bancos de Dados Dedutivos foi muito discutida no passado [BAN 86,MIY 90,RAM 92a]. Em função da forma como o Sistema DEDALO foi implementado, toda a otimização de consultas fica a cargo do Sistema Gerenciador de Banco de Dados que recebe a instrução SQL. Algumas otimizações adicionais, no entanto, poderiam ser realizadas pelo Sistema DEDALO, principalmente no processamento das consultas recursivas.

A implementação de raciocínio aproximado, penaliza excessivamente as aplicações que não necessitam desta facilidade. Uma nova versão do sistema, que não possuísse este tipo de raciocínio ou que possuísse mecanismos capazes de não penalizar as aplicações que não o utilizam, poderia ser criada. Além disso, a

capacidade de realizar raciocínio aproximado deve ser melhor explorada e testada, preferencialmente em uma aplicação real.

Aplicações típicas de Sistemas Especialistas freqüentemente necessitam que as conclusões atingidas sejam acompanhadas de uma explanação, ou seja, que o sistema informe como chegou a determinada conclusão. Este tema já foi abordado para Sistemas de Bancos de Dados Dedutivos [ARO 93, MAR 91], e seria uma interessante adição ao Sistema DEDALO.

Aplicações na área médica, como a descrita no 9, utilizam MLMs para representar suas regras. A implementação dos MLMs utilizando o sistema DEDALO mostrou ser bastante eficiente. No entanto, apenas os slots *data*, *logic* e *action* são tratados pelas regras de derivação. A construção de uma editor de MLMs que fosse capaz de armazenar os demais slots, traria uma grande contribuição para esta classe de aplicações.

# Anexo 1 BNF da Linguagem DEDALO

**Símbolos utilizados:**

< > Delimitam strings de caracteres que são os nomes de elementos sintáticos, os símbolos não terminais da linguagem.

::= O operador de definição. É usado em uma regra de produção para separar os elementos definidos pela regra de sua definição. Os elementos que estão sendo definidos aparecem no lado esquerdo do operador e a fórmula que define os elementos aparece à direita.

[ ] Indica elementos opcionais em uma fórmula. A porção da fórmula dentro dos colchetes pode ser explicitamente especificada ou pode ser omitida.

{ } Agrupam elementos em uma fórmula. A porção da fórmula dentro das chaves deve ser explicitamente especificada.

| O operador alternativo. Indica que a porção da fórmula seguindo a barra é uma alternativa para a porção precedendo a barra. Se a barra vertical aparece em uma posição onde não está envolvida em colchetes ou chaves, ela especifica uma alternativa completa para o elemento definido pela regra de produção. Se a barra vertical aparece em uma porção de uma fórmula envolvida em colchetes ou chaves, ela especifica alternativas para os conteúdos do par mais interno das chaves ou colchetes.

... Indicam que o elementos a que se aplicam em um fórmula podem ser repetidas qualquer número de vezes. Se as reticências aparecem imediatamente depois de uma chave '}', então ela se aplica à porção da fórmula entre a chave { e a chave }. Se as reticências aparecem depois de um outro elemento qualquer, então ele se aplica somente àquele elemento.

**BNF:**

<alteração> ::= <operação> <literal_positivo> [':-' <corpo> ] '.'
<operação> ::= 'ins' | 'upd' | 'del'
<exclusão> ::= 'del' <nome_relação> '.'
<regra> ::= <cabeça> ':-' <corpo> cf <número_real> '.'
<cabeça> ::= <nome_relação> '(' <campo> [@<digito>]
[ {',' <campo> [@<digito>] }... ] ')'
<nome_relação> ::= <identificador>
<campo> ::= <nome_campo> ':' <variável> | <constante> | <função> |
<expressão_aritmética>
<nome_campo> ::= <identificador>
<variável> ::= <identificador>
<constante> ::= {' " '{<letra> | <digito>}... ' " '} | <digito>...
<expressão_aritmética> ::= <variável> | <constante> {<operador> <variável> | <constante>}...
<letra> ::= 'A','B',...,'Z' | 'a',...,'z'
<digito> ::= '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'
<número_real> ::= { '0' '.' {'1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9'}...} | '1'
<corpo> ::= { <literal_positivo> | <literal_negativo> |
<expressão_de_comparação> } [ { ',' | ';' <literal_positivo> |
<literal_negativo> | <expressão_de_comparação> }... ] '.'
<literal_positivo> ::= <nome_relação> '(' <lista_de_argumentos> ')'
<literal_negativo> ::= 'not' '(' <literal_positivo> ')'
<lista_de_argumentos> ::= <argumento> [ {',' <argumento>}... ]
<argumento> ::= <nome_campo> ':' <variável> | <constante> |
<expressão_aritmética>
<expressão_de_comparação> ::= <variável> | ' " '<constante> ' " ' |
<expressão_aritmética> <operador_comparação> <variável> | ' " '
<constante> ' " ' | <expressão_aritmética>
<operador> ::= '+' | '-' | '*' | '/'
<operador_comparação> ::= '>' | '<' | '<=' | '>=' | '<>' | '='
<função> ::= <função_agregada> '(' <variável> ')'
<função_agregada> ::= 'avg' | 'count' | 'max' | 'min' | 'sum'
<identificador> ::= <letra> [ {<letra> | <dígito> }...]

# Bibliografia

[ABI 88] ABITEBOUL, Serge; HULL, Richard. Data Functions, DATALOG and Negation. In: ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1988, Chicago. **Proceedings...** New York:ACM Press, 1988. p. 143-153.

[ABI 90] ABITEBOUL, Serge. Towards a Deductive Object-oriented Database Language. **Data & Knowledge Engeneering,** Amsterdam, v. 5, n. 4, p. 263-287, Oct. 1990.

[ABI 91] ABITEBOUL, Serge; GRUMBACH, Stephane. A rule-based language with functions and sets. **ACM Transactions on Database Systems,** New York, v.16, n.1, p. 1-30, 1991.

[AGU 91] AGUIRRE, Anthony R.; RODERER, Nancy K. Managing Medical Logic Modules. In: SYMPOSIUM ON COMPUTER APPLICATIONS IN MEDICAL CARE, 13., 1991. **Proceedings...** New York:IEEE Computer Society Press, 1991.

[ALV 95] ALVES, M. Halfeld; LAURENT, D.; SPYRATOS, N. Update Transactions for Deductive Databases with Update Rules. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 10., 1995, Recife. **Anais...** Recife:UFPE/DI, 1995. p. 245-260.

[AME 92] AMERICAN NATIONAL STANDARDS INSTITUTE. **ANSI X3.135:1992, American National Standard for Information System - Database Language SQL**. [S.l.], 1992.

[APT 94] APT, R. Krzystof; BOL, Roland N. Logic Programming and Negation: A Survey. **The Journal of Logic Programming,** New York, v. 19/20, p. 9-71, May/July 1994.

[ARO 93] ARORA, Tarun et al. Explaining Program Execution in Deductive Systems. In: INTERNATIONAL CONFERENCE ON DEDUCTIVE AND OBJECT-ORIENTED DATABASES, 3., 1993, Phoenix, Arizona. **Proceedings...** Berlin:Springer-Verlag, 1993. p. 101-119. (Lecture Notes in Computer Science).

[ATZ 92] ATZENI, Paolo; TORLONE, Riccardo. Updating Relational Databases Through Weak Instance Interfaces. **ACM Transactions on Database Systems**, New York, v. 17, n. 4, p. 718-745, Dec. 1992.

[BAE 94] BARALIS, Elena; CERI, Stefano; PARABOSCHI, Stefano. Declarative Specification of Constraint Maintenance. In: INT. CONF. ON THE ENTITY RELATIONSHIP APPROACH (ER-94), 13., 1994. **Proceedings...** [S.l]:Springer-Verlag, 1994. p. 205-222. (Lecture Notes in Computer Science).

[BAN 86] BANCILHON, François; RAMAKRISHNAN, Raghu. An Amateur's Introduction to Recursive Query Processing Strategies. **SIGMOD Record,** New York, v. 15, n. 2, p. 16-52, June 1986. Trabalho apresentado na International Conference on Management of Data, 1986, Washington DC.

[BAR 94] BARJA, Maria L. et al. An Effective Deductive Object-Oriented Database Trough Language Integration. In: VLDB CONFERENCE, 20., 1994, Santiago, Chile. **Proceedings...** Hove:Morgan-Kaufmann, 1994. p.463- 474.

[BAY 93] BAYER, P.; LEFEBVRE, A.; VIEILLE L. **Architecture and Design of the EKS Deductive Database System.** [S.l.]:ECRC, 1993. (Technical Report).

[BLA 86] BLAKELEY, José A.; LARSON, Per-Ake; TOMPA, Frank.. Efficiently Updating Materialized Views. In: ACM SIGMOD 1986 INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, 1986, Washington D.C. **Proceedings...** New York:ACM Press, 1986. p.61-71.

[BÖH 94] BÖHLEN, Michael. **Managing Temporal Knowledge in Deductive Database.** Zürich: Swiss Federal Institute of Technology, 1994. PhD. Thesis.

[BOR 95] BORLAND INTERNATIONAL INC. **Delphi User's Guide.** [S.l:s.n.], 1995. 452p.

[BOR 95a] BORLAND INTERNATIONAL INC. **Delphi Component Writer's Guide.** [S.l:s.n.], 1995. 156p.

[BOR 95b] BORLAND INTERNATIONAL INC. **Delphi Database Application Developer's Guide.** [S.l:s.n.], 1995. 189p.

[BRY 88] BRY, F.; DECKER, H.; MANTHEY, R. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In: INTERNATIONAL CONFERENCE ON EXTENDING DATABASE TECHNOLOGY, 1988, Venice. **Proceedings...** Berlin:Springer-Verlag, 1988. p. 488-505.

[CAS 86] CASANOVA, Marco A. Uma Introdução a Bancos de Dados Dedutivos. In: SIMPÓSIO BRASILEIRO DE BANCOS DE DADOS, 1., 1986, Rio de Janeiro. **Anais...** Rio de Janeiro:PUC/SBC, 1986. p. 1-7.

[CER 90] CERI, Stefano; GOTTLOB, G.; TANCA, Letizia. **Logic Programming and Databases**. Berlin: Springer-Verlag, 1990. 284p. (Surveys in Computer Science).

[CER 90a] CERI, Stefano; WIDOM, Jennifer. Deriving Production Rules for Constraint Maintenance. In: VLDB CONFERENCE, 16., 1990, Brisbane, Australia. **Proceedings...** Palo Alto: Morgan Kaufmann, 1990. p. 566-577.

[CER 91] CERI, Stefano; WIDOM, Jennifer. Deriving Production Rules for Incremental View Maintenance. In: VLDB CONFERENCE, 17., 1991, Barcelona, Spain. **Proceedings...** San Mateo: Morgan Kaufmann, 1991. p. 577-589.

[CER 94] CERI, Stefano; WIDOW, Jennifer. Deriving incremental production rules for deductive data. **Information Systems**, New York, v. 19, n. 6, p. 467-490, Nov. 1994.

[CHA 93] CHAKRAVARTHY, Sharma. **A Comparative Evaluation of Active Relational Databases**. Gainesville: University of Florida, 1993. (Tech. Report UF-CIS-TR-93-002).

[CHI 90] CHIMENTI, D. et al. The LDL System prototype. **IEEE Transactions on Knowledge and Data Engineering**, New York, v. 2, n.1, p. 76-90, 1990.

[CON 93] CONSENS, Mariano; MENDELZON, Alberto. Hy: A hygraph-based query and visualization system. **SIGMOD Records**, New York, v.22, 2, p. 511-516, June 1993. Trabalho apresentado na ACM-SIGMOD International Conference On Management Of Data, 1993, Washington.

[COS 84] COSMADAKIS, Stavros S. Updates of Relational Views. **Journal of the Association for Computing Machinery,** v. 31, n. 4, p. 742-760, Oct.1984.

[DAY 94] DAYAL, Umeshwar; HANSON, Eric N.; WIDOM, Jennifer. Active Database System. In: Kim, Won (Ed.). **Modern Database Systems:** The Object Model, Interoperability, and Beyond. Massashussetts: Addison-Wesley, 1994.

[ELM 89] ELMASRI, Ramez; NAVATHE, Shamkant B. **Fundamentals of Database Systems**. Redwood: The Benjamin/Cummings, 1989. 751p.

[FRA 93] FRATERNALI, Piero; PARABOSCHI, Stefano. A Review of Repairing Techniques for Integrity Maintenance. In: INTERNATIONAL WORKSHOP ON RULES IN DATABASE SYSTEMS, 1., 1993, Edimburgh, Scotland. **Proceedings...** [S.l.]:Springer, 1994. p.333-346.

[FRE 91] FREITAG, B.; SCHÜTZ, H.; SPECHT, G. LOLA - a logic language for deductive databases and its implementation. In: INTERNATIONAL SYMPOSIUM ON DATABASE SYSTEMS FOR ADVANCED APPLICATIONS (DASFAA), 2., 1991, Tokyo, Japan. **Proceedings...** Singapore:World Scientific, 1992. p. 216-225.

[GAL 78] GALLAIRE, H.; MINKER, J.; NICOLAS, J.M.. (Eds.). **Logic and Databases**. New York:Plenum Press, 1978.

[GAL 84] GALLAIRE, Hervé; MINKER, Jack; NICOLAS, Jean-Marie. Logic and Databases: A Deductive Approach. **Computing Surveys**, New York, v.16, n. 2, p. 153-85, June 1984.

[GUP 92] GUPTA, Ashish; KATIYAR, Dinesh; MUMICK, Inderpal Singh. Counting Solutions to the View Maintenance Problem. In: WORKSHOP ON DEDUCTIVE DATABASES, JOINT INTERNATIONAL CONFERENCE AND SYMPOSIUM ON LOGIC PROGRAMMING, 1992, Washington DC. **Proceedings...** [S.l.:s.n.], 1992. p.185-194.

[GUP 93] GUPTA, Ashish; MUMICK, Inderpal S.; SUBRAHMANIAN, V.S. Maintaning views incrementally. **SIGMOD Records,** New York, v. 22, 2, p.157-166, June 1993. Trabalho apresentado na ACM SIGMOD International Conference On Management Of Data, 1993, Washington.

[HAR 92] HARRISON, John V.; DIETRICH, Suzanne. Maintenance of materialized views in a deductive database: An update propagation approach. In: WORKSHOP ON DEDUCTIVE DATABASES, JOINT INTERNATIONAL CONFERENCE AND SYMPOSIUM ON LOGIC PROGRAMMING, 1992, Washington. **Proceedings...** [S.l:s.n.], 1992. p 56-65.

[HRI 90] HRIPESAK, George et al. The Arden Sintax for Medical Logic Modules. In: SYMPOSIUM ON COMPUTER APPLICATIONS IN MEDICAL CARE, 14., 1990, Washington. **Proceedings...** New York: IEEE Computer Society Press, 1990. p. 200-204.

[JAR 94] JARKE, Matthias. **ConceptBase Tutorial**. 1994. Aachen, Germany. (Disponível via WWW em http://www.informatik.rwth-aachen.de/I5/CBdoc/cbflyer.html).

[JEU 93] JEUSFELD, M., STAUDT, M. Query optmization in deductive object bases. In: **Query Processing for Advanced Database Aplications.** [S.l.]:Morgan-Kaufmann, 1993.

[KAK 90] KAKAS, A.C.; MANCARELLA, P.. Database Updates Trough Abduction. In: VLDB CONFERENCE, 16., 1990, Brisbase, Austrália. **Proceedings...** Palo Alto:Morgan-Kaufmann, 1990. p. 650-661.

[KIE 90] KIERNAN, Gérald; MAINDREVILLE, Christophe de; SIMON, Eric. **Making Deductive Database A Pratical Technology:** A Step Forward. Rocquencourt: Institut National de Recherche en Informatique et en Automatique, INRIA-Rocquencourt, 1990. Janvier 1990. (Rapport de Recherche No. 1153).

[KIE 93] KIEBLING, Werner; SCHIMIDT, H. **DECLARE and SDS:** Early efforts to commercialize deductive database technology. 1993. (Submitted).

[KOR 94] KORTH, Henry F.; SILBERCHATZ, Abraham. Sistemas Relacionais Estendidos. In: **Sistema de Bancos de Dados.** 2.ed. São Paulo: Makron Books, 1994. p. 481-497.

[KOW 87] KOWALSKI, R.; SADRI, F.; SOPER, P. Integrity Checking in Deductive Databases. In: VERY LARGE DATABASES CONFERENCE, 13., 1987, Brighton. **Proceedings...** Los Altos:Morgan-Kaufmann, 1987. p. 61-69.

[LAN 90] LANGERAK, Rom. View Updates in Relational Databases with an Independent Scheme. **ACM Transactions on Database Systems**, New York, v. 15, n. 1, p. 40-66, Mar. 1990.

[LAU 94] LAURENT, Dominique; SPYRATOS, Nicolas. A Partition Model Approach to Updating Universal Scheme Interfaces. **IEEE Transactions on Knowledge and Data Engineering**, New York, v. 6, n. 2, p. 316-330, Apr. 1994.

[LOB 93] LOBO, Jorge. **A Tutorial on Deductive Databases**. Palestra proferida em 15 de Outubro de 1993 na XIII International Conference of the Chilean Computer Society.

[MAR 90] MARTI, R.; WIELAND, C. Implementing Fuzzy Logic for a Deductive Database Manager. In: INTERNATIONAL WORKSHOP OF INTEGRATED INTELLIGENT INFORMATION SYSTEMS, 1990, Tuczuno Castle, Portland. **Proceedings...** [S.l.:s.n.], 1990. p. 235-247.

[MAR 91] MARTI, R. Research in Deductive Databases et ETH: The LogiQuel Project. In: DATABASE RESEARCH IN SWITZERLAND, 1991. **Proceedings...** [S.l.:s.n.], 1991. p.130-143.

[MAT 89] MATTOS, Nelson Mendonça. **An Approach to Knowledge Base Management**. Berlin: Springer Verlag, 1989. (Lecture Notes in Computer Science).

[MIY 90] MIYAZAKI, Nobuyoshi. Selection Propagation in Deductive Databases. **Data & Knowledge Engineering,** Amsterdam, v. 5, n. 4, p. 313-331, Oct. 1990.

[MOE 88] MOERKOTTE, Guido; KARL, Stefan. Efficient Consistency Control in Deductive Database. In: INTERNATIONAL CONFERENCE ON DATABASE THEORY, 2., 1988, Bruges, Belgium. **Proceedings...** [S.l]:Springer-Verlag, 1988. p. 118-128. (Lecture Notes in Computer Science).

[MOE 91] MOERKOTTE, Guido; LOCKEMANN, Peter. Reactive Consistency Control in Deductive Databases. **ACM Transactions on Database Systems**, New York, v. 16, n. 4, p. 670-702, Dec. 1991.

[MON 93] MONTESI, Danilo. **A Model for Updates and Transactions in Deductive Databases.** Pisa: Dipartamento di Informatica, Università di Pisa, 1993. PhD Thesis.

[MOR 86] MORRIS, Katherine; ULLMAN, Jeffrey; GELDER, Allen van. Design Overview of the NAIL! System. In: INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING, 3., 1986, New York. **Proceedings...** [S.l.]:Springer, 1986. p.554-568. (Lecture Notes in Computer Science).

[NAV 89] NAVÓN, Jaime. De las Bases de Datos Relacionales de los 80's a las Bases de Datos Deductivas de los 90's. In:**Apuntes de Ingeniería**. Santiago:PUCC, 1989. p. 21-34.

[NEV 95] NEVES, Eugênio Rodrigo Zimmer. **Desenvolvimento de um protótipo de sistema de suporte a decisão baseado em alertas vinculado a um sistema de informações sobre medicamentos cardiovasculares.** Porto Alegre: Faculdade de Farmácia, Universidade Federal do Rio Grande do Sul, 1995. Dissertação de Mestrado.

[OLI 91] OLIVÉ, Antoni. Integrity Constraints Checking in Deductive Databases. In: INTERNATIONAL CONFERENCE ON VERY LARGE DATABASES, 17., 1991, Barcelona. **Proceedings...** San Mateo:Morgan-Kaufmann, 1991. p. 513-523.

[PAR 94] PARABOSCHI, Stefano. **A Taxonomy of Methods for the Incremental Maintenance of Materialized Views.** Milano: Politecnico di Milano, 1994. (Technical Report IDEA.DD.22.P004).

[PHI 91] PHIPPS, Geoffrey; DERR, Marcia A.; ROSS, Kenneth A. Glue-Nail: A Deductive Database System. **SIGMOD Records,** New York, v. 20, 2, p.308-317, June 1991. Trabalho apresentado na ACM SIGMOD International Conference on Management of Data, 1991, Denver.

[PRZ 88] PRZYMUSINSKI, T. C. On the declarative semantics of stratified deductive databases. In: **Foundations of Deductive and Logic Programming**. [S.l.]:Morgan-Kaufmann, 1988. p. 193-216.

[RAM 92] RAMAKRISHMAN, Raghu; SRIVASTAVA, Divesh; SUDARSHAN, S. CORAL - Control, Relations and Logic. In: VLDB CONFERENCE, 18., 1992, Vancouver, Canada. **Proceedings...** San Mateo:Morgan-Kaufmann, 1992. p. 238 a 250.

[RAM 92a] RAMAKRISHNAN, Raghu; SRIVASTAVA, Divesh; SUDARSHAN, S. Controlling the Search in Bottom-Up Evaluation. In: JOINT INTERNATIONAL CONFERENCE AND SYMPOSIUM ON LOGIC PROGRAMMING, 1992, Washington, SC. **Proceedings...** [S.l.:s.n.], 1992.

[RAM 95] RAMAKRISHNAN, Raghu; ULLMAN, Jeffrey D. A Survey of Research on Deductive Database System. **Journal of Logic Programming**, New York, p. 125-149, May 1995.

[RUB 95] RUBENKING, Neil J. **Programação em Delphi para leigos.** São Paulo: Berkeley Brasil, 1995. 372p.

[SHA 88] SHAO, J.; BELL, D. A.; HULL, M.E.C. LQL: A Unified Language For Deductive Database Systems. In: IFIP Working Conference on the Role of Artificial Intelligence in Databases and Information Systems, 1988, Guangzhou, China. **Proceedings...** Amsterdam:North-Holland, 1990. p. 329 - 343.

[SIM 92] SIMON, E.; KIERNAN, J.; MAINDREVILLE, C. Implementing High Level Active Rules on top of a Relational DBMS. In: VLDB CONFERENCE, 18., Vancouver, Canada. **Proceedings...** San Mateo:Morgan Kaufmann, 1992. p. 315-326.

[STO 90] STONEBRAKER, Michael; JHINGRAN, Anant; GOH, Jeffrey; POTAMIANOS, Spyros. On Rules, Procedures, Caching and Views in Data Base Systems. **SIGMOD Records,** New York, v. 19, 2, June, 1990. Trabalho apresentado na ACM SIGMOD International Conference on Management of Data, 1990, Atlantic City, NJ.

[TAN 95] TANAKA, Asterio. Bancos de Dados Ativos. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 10., 1995, Recife. **Anais...** Recife: UFPE/DI, 1995. p. 29-53.

[TOM 88] TOMASIC, Anthony. View Update Translation via Deduction and Annotation. In: INTERNATIONAL CONFERENCE ON DATABASE THEORY, 2., 1988, Bruges, Belgium. **Proceedings...** [S.l.]:Springer-Verlag, 1988. (Lecture Notes in Computer Science).

[TOM 93] TOMASIC, Anthony. **Correct View Update Translations via Containment**. Stanford: Stanford University, 1993. (Science Technical Note STAN//CS-TN-94-3).

[TOR 91] TORLONE, Riccardo; ATZENI, Paolo. Updating Deductive Databases with Functional Dependences. In: INTERNATIONAL CONFERENCE OF DEDUCTIVE AND OBJECT-ORIENTED DATABASES, 2., 1991, Munich, Germany. **Proceedings...** Berlin:Springer-Verlag, 1991. (Lecture Notes in Computer Science).

[TSU 91] TSUR, Shalom. Deductive Databases in Action. In: ACM-SIGACT-SIGMOD-SICART SYMPOSIUM ON PRINCIPLES OF DATABASE SYSTEMS, 10., 1991, Denver-Colorado. **Proceedings...** New York:ACM Press, 1991. p. 142-152.

[ULL 85] ULLMAN, Jeffrey D. Implementation of Logical Query Languages for Databases. **ACM Transactions on Database Systems**, New York, v. 10, n. 3, p. 289-321, Sept. 1985.

[ULL 91] ULLMAN, Jeffrey D. A Comparison Between Deductive and Object-Oriented Database Systems. In: INTERNATIONAL CONFERENCE IN DEDUCTIVE AND OBJECT-ORIENTED DATABASES, 2., 1991, Munich, Germany. **Proceedings...** Berlin:Springer-Verlag, 1991. p. 263-277. (Lecture Notes in Computer Science).

[VAG 91] VAGHANI, Jayen et al. Design Overview of the Aditi deductive database system. In: INTERNATIONAL CONFERENCE ON DATA ENGINEERING, 7., 1991, Kobe, Japan. **Proceedings...** Los Alamitos:IEEE Computer Society Press, 1991. p.240-247.

[VIE 86] VIEILLE, Laurent. Recursive axioms in deductive databases: The query-subquery approach. In: INTERNATIONAL CONFERENCE ON EXPERT DATABASE SYSTEMS, 1., 1986, Charleston, South Carolina. **Proceedings...** [S.l.]:Benjamin Cummings, 1987. p. 179-193.

[VIE 87] VIEILLE, Laurent. Database complete proof procedures based on SLD-resolution. In: INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING, 4., 1987, Melbourne. **Proceedings...** [S.l.]:MIT Press, 1987. p. 74-103.

[VIO 93] VIOT, Greg. Fuzzy Logic: Concepts to Constructs. **AI Expert**, San Francisco, CA, p.26-33, Nov. 1993.

[ZAD 65] ZADEH, L. A. Fuzzy Sets. **Information and Control**, New York, v. 8, 1965.