Projeto de Regras Confluentes para Bancos de Dados Ativos Usando Tabelas de Decisão

Mabelly de Oliveira Furtado

Dissertação submetida em cumprimento parcial dos
requisitos para obtenção do grau de

Mestre em Ciência da Computação

Orientador: **Prof. Dr. Marcelo Finger**

(A autora recebeu apoio financeiro do CNPq durante a elaboração deste trabalho)

Instituto de Matemática e Estatística da USP

- São Paulo, dezembro de 1999 -

Projeto de Regras Confluentes para Bancos de Dados Ativos Usando Tabelas de Decisão

Este exemplar corresponde à versão final, devidamente corrigida, da dissertação apresentada por Mabelly de Oliveira Furtado e aprovada pela comissão julgadora.

São Paulo, dezembro de 1999.

Banca Examinadora:

Prof. Dr. Marcelo Finger (orientador)

IME-USP

Prof. Dr. Astério Tanaka

UNI-RIO

Prof. Dr. João Eduardo Ferreira

IME-USP

Projeto de Regras Confluentes para Banco de Dados Ativos Usando Tabelas de Decisão

por Mabelly de Oliveira Furtado

Resumo

Esta dissertação propõe um novo método de projeto de regras confluentes para bancos de dados ativos.

Nos sistemas de bancos de dados ativos difundidos no mercado e na literatura, regras são projetadas sem a preocupação com seu inter-relacionamento e, na maioria das vezes, com sua confluência. No primeiro caso, o desenvolvedor de aplicação é, com frequência, incapaz de ter uma visão global das regras já projetadas e, conseqüentemente, de saber se algumas propriedades vitais aos bancos de dados ativos são satisfeitas ou não. Já no segundo caso, o sistema de gerenciamento de banco de dados ativo não garante a consistência das informações por ele gerenciadas.

Neste trabalho adaptaremos as tabelas de decisão às regras que caracterizam os bancos de dados ativos e apresentaremos um projeto de semântica de regras confluentes.

Confluent Rules Design for Active Database Using Decision Tables

by Mabelly de Oliveira Furtado

Abstract

This dissertation proposes a new method of confluent rules design for active databases.

In active databases systems diffused in the market and in the literature, rules are designed without the concern with their inter-relationship and, most of the time, with their confluence. In the first case, the application developer is, frequently, unable to already have a global vision of the rules designed and, consequently, of knowing some vital properties to the active databases are satisfied or not. Already in the second case, the active database manager system doesn't guarantee the consistency of the information for its manageable.

In this work we will adapt the decision tables to the rules, that characterize the active databases, and we will present a design of semantics of confluent rules.

A Deus.

AGRADECIMENTOS

Ao Prof. Dr. Marcelo Finger, pela inestimável orientação e paciência demonstrados durante a realização deste trabalho.

Aos funcionários da Biblioteca "Carlos Benjamin de Lyra" do Instituto de Matemática e Estatística da USP, pela colaboração e paciência com que me forneceram as informações necessárias à elaboração do estudo de caso apresentado no capítulo 5.

À minha família, por todo incentivo, apoio e carinho. Em especial ao meu marido, Marcílio Xavier Furtado, pelo companheirismo e paciência que demonstrou durante o desenvolvimento deste trabalho.

Ao CNPq, pelo apoio financeiro concedido.

Enfim, agradecer a todos que, direta ou indiretamente, participaram do momento especial de realização deste trabalho.

ÍNDICE

INTRODUÇÃO

1.1 ORIGEM DOS BANCOS DE DADOS ATIVOS	1
1.2 PRINCIPAIS PROBLEMAS	5
1.3 UTILIZAÇÃO DOS BANCOS DE DADOS ATIVOS.....	6
1.3.1 Aplicações Internas	6
1.3.2 Aplicações Estendidas	7
1.3.3 Aplicações Externas	7
1.4 OBJETIVOS DESTA DISSERTAÇÃO	7

BANCOS DE DADOS ATIVOS

2.1 O QUE SÃO OS BANCOS DE DADOS ATIVOS ?	9
2.2 REGRAS INTENCIONAIS	9
2.2.1 Evento	10
2.2.2 Condição	10
2.2.3 Ação	11
2.2.4 Outras Características da Linguagem de Regras.....	12
2.2.4.1 Valores de Transição	12
2.2.4.2 Prioridade entre Regras Intencionais	13
2.3 SEMÂNTICA DE EXECUÇÃO DE REGRAS INTENCIONAIS	13
2.3.1 Dependência entre as Partes de uma Regra Intencional.....	13
2.3.2 Modelo Fundamental.....	14
2.3.3 Considerações a Respeito da Semântica de Execução de Regras Intencionais.....	14
2.3.3.1 Granularidade do Processamento de Regras Intencionais	15
2.3.3.2 Execução Orientada à Instância versus Orientada a Conjunto	16
2.3.3.2.1 Execução Orientada à Instância.....	16
2.3.3.2.2 Execução Orientada a Conjunto.....	16
2.3.3.3 Algoritmos Iterativos versus Algoritmos Recursivos	17
2.3.3.3.1 Algoritmos Iterativos.....	17
2.3.3.3.2 Algoritmos Recursivos.....	17
2.3.3.4 Resolução de Conflitos	17
2.3.3.5 Execução Sequencial versus Execução Concorrente	18
2.3.3.5.1 Execução Sequencial.....	18
2.3.3.5.2 Execução Concorrente.....	18
2.3.3.6 Formas de Acoplamento	18
2.3.3.7 Terminação	19
2.3.3.8 Confluência.....	19
2.3.3.9 Determinismo Observável	20
2.4 ARQUITETURA DOS SISTEMAS DE BANCOS DE DADOS ATIVOS.....	20
2.4.1 Arquitetura em Camadas.....	20
2.4.2 Arquitetura Embutida.....	21
2.4.3 Arquitetura Compilada.....	22
2.5 COMPARAÇÃO ENTRE O POSTGRES, O ARIEL, O STARBURST E A EXTENSÃO “ATIVA” PARA BANCOS DE DADOS RELACIONAIS DESTA DISSERTAÇÃO.....	23

CONFLUÊNCIA DE REGRAS INTENCIONAIS

3.1 O QUE É UM CONJUNTO DE REGRAS INTENCIONAIS CONFLUENTES?	25
3.2 SOLUÇÕES PARA O PROBLEMA DA CONFLUÊNCIA	25
3.2.1 <i>Análise Estática</i>	26
3.2.1.1 Grafo de Execução de Regras Intencionais	26
3.2.1.1.1 Definição	26
3.2.1.1.2 Propriedades	27
3.2.1.2 Comutatividade entre Regras	27
3.2.1.3 Método para Análise da Confluência	29
3.2.1.4 Considerações	31
3.2.2 <i>Modificação da Semântica de Execução de Regras Intencionais</i>	32
3.2.2.1 Conjunto de Ações Consistentes	32
3.2.2.2 Semântica Confluente	33

TABELAS DE DECISÃO COMO FORMA DE REPRESENTAR REGRAS INTENCIONAIS

4.1 TABELAS DE DECISÃO PARA BANCOS DE DADOS ATIVOS	35
4.1.1 <i>O Que é uma Tabela de Decisão para Banco de Dados Ativos?</i>	35
4.1.2 <i>Interpretação e Estrutura de uma TDBDA</i>	36
4.1.3 <i>Outras Características</i>	39
4.1.4 <i>Compleitude de uma TDBDA</i>	41
4.1.5 <i>Desmembramento e Aglutinação de TDBDA's</i>	41
4.1.5.1 Desmembramento de uma TDBDA	42
4.1.5.2 Aglutinação de uma TDBDA	43
4.1.6 <i>TRADUÇÃO DE TDBDA'S PARA GATILHOS DE BANCO DE DADOS</i>	44
4.2 VANTAGENS DO USO DE TDBDA'S SOBRE O MÉTODO TRADICIONAL DE PROJETO DE REGRAS INTENCIONAIS	45

ESTUDO DE CASO: MODELAGEM DA BIBLIOTECA DO IME/USP

5.1 INTRODUÇÃO	47
5.2 DESCRIÇÃO GERAL DO SISTEMA	47
5.3 PRINCIPAIS REGRAS DE NEGÓCIO	47
5.4 O DIAGRAMA DE ENTIDADE-RELACIONAMENTO DA BIBLIOTECA DO IME	49
5.5 O ESQUEMA DE BANCO DE DADOS DA BIBLIOTECA DO IME	49
5.6 REPRESENTAÇÃO DAS REGRAS DE NEGÓCIO ATRAVÉS DE TDBDA'S	50
5.6.1 <i>Regras de Negócio para a Consulta a Livros</i>	50
5.6.2 <i>Regras de Negócio para a Consulta a Periódicos</i>	52
5.6.3 <i>Regras de Negócio para a Reserva de Livros</i>	52
5.6.4 <i>Regras de Negócio para a Devolução de Livros</i>	53
5.6.5 <i>Regras de Negócio para a Devolução de Livros Consultados</i>	53
5.6.6 <i>Regras de Negócio para o Empréstimo de Livros</i>	54
5.7. UM EXEMPLO DE DESMEMBRAMENTO E AGLUTINAÇÃO DE TDBDA'S	59
5.7.1 <i>Desmembramento de uma TDBDA</i>	59
5.7.2 <i>Aglutinação das TDBDA's</i>	62
5.7.3 <i>Condições sobre Aglutinação e Desmembramento de TDBDA's: uma Experiência</i>	73

CONCLUSÃO

APÊNDICE A: UMA LINGUAGEM PARA BANCO DE DADOS ATIVOS

A.1 LINGUAGEM DE CONSULTA	76
A.1.1 Linguagem de Manipulação de Dados (DML).....	77
A.1.2 Controle de Transações	80
A.2 LINGUAGEM DE PROGRAMAÇÃO	80
A.2.1 Bloco PL/SQL.....	81
A.2.2 Variáveis e Constantes.....	81
A.2.3 Tipos de Dados.....	83
A.2.4 Estruturas de Controle.....	84
A.2.4.1 Condição.....	85
A.2.4.2 Iteração	86
A.2.5 Subprogramas	88
A.2.6 Interagindo com o Oracle	90
A.3 LINGUAGEM DE REGRAS INTENCIONAIS	91
A.3.1 O Comando CREATE RULE.....	92
A.3.2 O Comando DROP RULE.....	93
A.3.3 O Comando ALTER RULE.....	93
A.3.4 Outras Características	93

APÊNDICE B: TABELAS DE DECISÃO

B.1 REGRAS DE DECISÃO	96
B.2 O QUE É UMA TABELA DE DECISÃO?.....	97
B.3 ESTRUTURA E INTERPRETAÇÃO DAS TABELAS DE DECISÃO	97
B.4 OUTRAS CARACTERÍSTICAS.....	99
B.4.1 Tabelas de Decisão Estendidas e Mistas.....	99
B.4.2 Condições Dependentes e Independentes.....	100
B.4.3 Os Símbolos '\$', '*' e '-'.....	101
B.4.4 "Else Rule"	102
B.4.5 Regras e Tabelas de Decisão Ambíguas	102
B.5 COMPLETUDE DE UMA TABELA DE DECISÃO	103
B.6 DIVISÃO E FATORAÇÃO DE TABELAS DE DECISÃO.....	104
B.6.1 Divisão de Tabelas de Decisão	105
B.6.2 Fatoração de Tabelas de Decisão por uma Linha de Condição	106
B.7 TRADUÇÃO DE TABELAS DE DECISÃO	107
B.7.2 "Switch Method"	108
B.7.3 Método de Máscaras	109
B.7.4 Método de Fatorações Sucessivas.....	111

APÊNDICE C: AS TABELAS DE DECISÃO PARA BANCO DE DADOS ATIVOS DO ESTUDO DE CASO

C.1 REPRESENTAÇÃO DAS REGRAS DE NEGÓCIO ATRAVÉS DE TDBDA'S.....	113
---	-----

<i>C.1.1 Regras de Negócio para a Consulta a Livros</i>	<i>113</i>
<i>C.1.2 Regras de Negócio para a Consulta a Periódicos</i>	<i>114</i>
<i>C.1.3 Regras de Negócio para a Reserva de Livros</i>	<i>116</i>
<i>C.1.4 Regras de Negócio para a Devolução de Livros.....</i>	<i>116</i>
<i>C.1.5 Regras de Negócio para a Devolução de Livros Consultados.....</i>	<i>118</i>
<i>C.5.6. Regras de Negócio para o Empréstimo de Livros.....</i>	<i>121</i>

BIBLIOGRAFIA

ÍNDICE DE FIGURAS

<i>Figura 1.1 - Exemplo do processamento em Bancos de Dados Passivos.....</i>	<i>4</i>
<i>Figura 1.2 - Exemplo do processamento em Bancos de Dados Ativos.....</i>	<i>5</i>
<i>Figura 2.1 - Grafo de disparo com ciclo [WC96].....</i>	<i>19</i>
<i>Figura 2.2 - Banco de Dados Ativo cuja arquitetura é em camadas.....</i>	<i>21</i>
<i>Figura 2.3 - Banco de Dados Ativo cuja arquitetura é embutida, adaptado de [KA93].....</i>	<i>22</i>
<i>Figura 3.1 - Exemplo de um grafo de execução de regras intencionais.....</i>	<i>26</i>
<i>Figura 3.2 - Regras comutativas.....</i>	<i>28</i>
<i>Figura 3.3 - Confluência de regras intencionais.....</i>	<i>29</i>
<i>Figura 3.4 - Grafo de execução representando parte dos caminhos p_1 e p_2.....</i>	<i>30</i>
<i>Figura 3.5 - Grafo de execução cujos caminhos p_1 e p_2 convergem para o mesmo estado final.....</i>	<i>30</i>
<i>Figura 4.1 - Estrutura de representação de uma TDBDA.....</i>	<i>36</i>
<i>Figura 4.2 - Linha de evento de uma TDBDA.....</i>	<i>37</i>
<i>Figura 4.3 - Linhas de condição de uma TDBDA.....</i>	<i>37</i>
<i>Figura 4.4 - Linhas de ação de uma TDBDA.....</i>	<i>37</i>
<i>Figura 4.5 - Descrição das condições de uma TDBDA.....</i>	<i>37</i>
<i>Figura 4.6 - Entradas de condições de uma TDBDA.....</i>	<i>38</i>
<i>Figura 4.7 - Descrição das ações de uma TDBDA.....</i>	<i>38</i>
<i>Figura 4.8 - Entradas de ação de uma TDBDA.....</i>	<i>38</i>
<i>Figura 4.9 - Coluna de entradas de condição de uma TDBDA.....</i>	<i>39</i>
<i>Figura 4.10 - Coluna de entradas de ação de uma TDBDA.....</i>	<i>39</i>
<i>Figura 4.11 - Exemplo de uma TDBDA limitada.....</i>	<i>40</i>
<i>Figura 4.12 - Exemplo de TDBDA estendida.</i>	<i>40</i>
<i>Figura 4.13 - Exemplo de uma TDBDA mista.....</i>	<i>40</i>
<i>Figura 4.14 - Exemplo de uma TDBDA que utiliza o símbolo '-'.....</i>	<i>40</i>
<i>Figura 4.15 - TDBDA da figura 4.14 sem o símbolo '-'.....</i>	<i>41</i>
<i>Figura 4.16 - TDBDA da figura 4.1 modificada, com duas regras ambíguas.....</i>	<i>41</i>
<i>Figura 4.17 - TDBDA que trata do empréstimo de livros a leitores do IME.....</i>	<i>42</i>
<i>Figura 4.18 - TDBDA's resultantes do desmembramento da TDBDA da figura 4.17.....</i>	<i>43</i>
<i>Figura 4.19 - TDBDA resultante da aplicação da primeira etapa do processo de aglutinação das TDBDA's da figura 4.18.....</i>	<i>43</i>
<i>Figura 4.20 - TDBDA resultante da aplicação da segunda etapa do processo de aglutinação das TDBDA's da figura 4.18.....</i>	<i>44</i>
<i>Figura 5.1 - TDBDA com as regras de negócio referentes à consulta a livros.....</i>	<i>50</i>
<i>Figura 5.2 - Diagrama E-R para Controle Bibliotecário da Biblioteca do IME.....</i>	<i>51</i>
<i>Figura 5.3 - TDBDA com as regras de negócio referentes à consulta a periódicos.....</i>	<i>52</i>

<i>Figura 5.4 - TDBDA com as regras de negócio referentes à Reserva de Livros.....</i>	<i>52</i>
<i>Figura 5.5 - TDBDA com as regras de negócio referentes à devolução de livros emprestados aos leitores da Biblioteca do IME.....</i>	<i>53</i>
<i>Figura 5.6 - TDBDA com as regras de negócio referentes à devolução de livros consultados pelos leitores.....</i>	<i>53</i>
<i>Figura 5.7 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME.....</i>	<i>54</i>
<i>Figura 5.8 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Continuação)..</i>	<i>55</i>
<i>Figura 5.9 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Continuação)..</i>	<i>56</i>
<i>Figura 5.10 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Continuação)..</i>	<i>57</i>
<i>Figura 5.11 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Continuação)..</i>	<i>58</i>
<i>Figura 5.12 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IME.....</i>	<i>60</i>
<i>Figura 5.13 -TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IME.....</i>	<i>60</i>
<i>Figura 5.14 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IME.....</i>	<i>60</i>
<i>Figura 5.15 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IME.....</i>	<i>61</i>
<i>Figura 5.16 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IME.....</i>	<i>61</i>
<i>Figura 5.17 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IME.....</i>	<i>61</i>
<i>Figura 5.18 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12 e 5.13.....</i>	<i>62</i>
<i>Figura 5.19 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13 e 5.14.....</i>	<i>62</i>
<i>Figura 5.20 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15....</i>	<i>63</i>
<i>Figura 5.21 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 (continuação).....</i>	<i>63</i>
<i>Figura 5.22 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 (continuação).....</i>	<i>64</i>
<i>Figura 5.23 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16).....</i>	<i>65</i>
<i>Figura 5.24 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 (continuação).....</i>	<i>66</i>
<i>Figura 5.25 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 (continuação).....</i>	<i>67</i>
<i>Figura 5.26 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 (continuação)).....</i>	<i>68</i>

<i>Figura 5.27 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 e 5.17.....</i>	<i>69</i>
<i>Figura 5.28 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 e 5.17 (Continuação).....</i>	<i>70</i>
<i>Figura 5.29 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 e 5.17 (Continuação).....</i>	<i>71</i>
<i>Figura 5.30 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 e 5.17 (Continuação).....</i>	<i>72</i>
<i>Figura A.1 - Estrutura de um bloco PL/SQL [Orc96].....</i>	<i>81</i>
<i>Figura A.2 - Principais tipos de dados escalares da PL/SQL.....</i>	<i>83</i>
<i>Figura A.3 - Tipos de dados compostos da PL/SQL.....</i>	<i>84</i>
<i>Figura A.4 - Representação da estrutura de controle seqüencial [Orc96].....</i>	<i>85</i>
<i>Figura A.5 - Representações das estruturas de controle condição e iteração, respectivamente [Orc96].....</i>	<i>85</i>
<i>Figura B.1 - Uma das formas de representarmos as tabelas de decisão.....</i>	<i>97</i>
<i>Figura B.2 - Linhas de condição e ação de uma tabela de decisão.....</i>	<i>98</i>
<i>Figura B.3 - Descrições e entradas de condições e ações de uma tabela de decisão.....</i>	<i>98</i>
<i>Figura B.4 - Exemplo de uma tabela de decisão estendida.....</i>	<i>99</i>
<i>Figura B.5 - Exemplos de tabela de decisão mista.....</i>	<i>100</i>
<i>Figura B.6 - Exemplo de condições explicitamente dependentes.....</i>	<i>100</i>
<i>Figura B.7 - Exemplo de condições implicitamente dependentes.....</i>	<i>100</i>
<i>Figura B.8 - Exemplo de utilização dos símbolos '\$' e '*'.....</i>	<i>101</i>
<i>Figura B.9 - Exemplo de utilização do símbolo '-'.....</i>	<i>102</i>
<i>Figura B.10 - Exemplo de "else rule".....</i>	<i>102</i>
<i>Figura B.11 - Exemplo de Tabela de Decisão Ambígua.....</i>	<i>102</i>
<i>Figura B.12 - Exemplo de Tabela de Decisão Completa.....</i>	<i>103</i>
<i>Figura B.13 - Número de Regras de uma Tabela de Decisão que Utiliza o Símbolo '-'.....</i>	<i>104</i>
<i>Figura B.14 - Uso da Regra "Else".....</i>	<i>104</i>
<i>Figura B.15 - Tabela de Decisão do Exemplo do Início do Capítulo.....</i>	<i>105</i>
<i>Figura B.16 - Tabela de Decisão T2V, resultante da divisão da tabela da figura B.15.....</i>	<i>106</i>
<i>Figura B.17 - Tabela de Decisão T2F, resultante da divisão da tabela da figura B.15.....</i>	<i>106</i>
<i>Figura B.18 - Tabela de Decisão T'.....</i>	<i>106</i>
<i>Figura B.19 - Forma usual de representarmos a Fatoração de Tabelas de Decisão.....</i>	<i>107</i>
<i>Figura C.1 - TDBDA com as regras de negócio referentes à consulta a livros.....</i>	<i>113</i>
<i>Figura C.2 - TDBDA com as regras de negócio referentes à consulta a periódicos.....</i>	<i>114</i>
<i>Figura C.3 - TDBDA com as regras de negócio referentes à Reserva de Livros.....</i>	<i>116</i>
<i>Figura C.4 - TDBDA com as regras de negócio referentes à devolução de livros emprestados aos leitores da Biblioteca do IME.....</i>	<i>117</i>

<i>Figura C.5 - TDBDA com as regras de negócio referentes à devolução de livros consultados pelos leitores.....</i>	<i>118</i>
<i>Figura C.6 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME.....</i>	<i>121</i>
<i>Figura C.7 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Continuação).122</i>	
<i>Figura C.8 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Continuação).123</i>	
<i>Figura C.9 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Continuação).124</i>	
<i>Figura C.10 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Cont.).</i>	<i>125</i>

INTRODUÇÃO

1.1 Origem dos Bancos de Dados Ativos

O armazenamento de dados evoluiu consideravelmente nas últimas décadas. Passamos do tradicional acesso seqüencial a arquivos aos sistemas gerenciadores de bancos de dados. Tais sistemas trouxeram benefícios consideráveis ao armazenamento e acesso de dados, tais como:

- independência dos programas de aplicação em relação à estrutura ou disposição física dos dados, acarretando um menor impacto nas manutenções; e
- redução do nível de redundância, o que levou ao melhor aproveitamento do espaço de armazenamento, menor probabilidade de inconsistências devidas a dados conflitantes e simplificação do processo de desenvolvimento.

Esses sistemas são conhecidos como **Sistemas de Bancos de Dados Convencionais** ou **Passivos**, pois neles os dados somente são criados, recuperados, modificados e excluídos em resposta às operações emitidas pelos usuários ou aplicações [WC96].

À medida que os bancos de dados passivos começaram a ser largamente utilizados em várias áreas da informática, problemas foram surgindo e suas limitações ficaram evidentes. As principais limitações são as seguintes:

- *implementação de regras de negócio¹ e gatilhos*: os bancos de dados passivos suportam restrições de integridade simples, tais como: valores únicos (o código de um livro é um atributo que o identifica entre vários livros, portanto não faz sentido termos dois ou mais livros com o mesmo código na relação livros), valores não nulos (consideremos o título de um livro, não faz sentido termos o código, o ano de publicação e a editora de

¹ São restrições de integridade complexas intimamente relacionadas com o domínio da aplicação. Exemplo: numa loja de doces, um pedido ao fornecedor de chocolate branco é feito quando o estoque fica com vinte caixas ou menos do produto.

um livro sem o título do mesmo), itens de dados com referência a outros itens (sejam o autor de um livro armazenado numa relação autores, esta relação deverá conter dados que permitam extrair as informações dos livros escritos por este autor), entre outros; não sendo, portanto, capazes de implementar diretamente as regras de negócio e os gatilhos. Há duas soluções para esta limitação. Na primeira solução, o desenvolvedor implementará as regras de negócio na própria aplicação, isto é, as aplicações farão as validações apropriadas quando estiver inserindo, atualizando ou excluindo dados do banco de dados. Esta solução é responsável por alguns problemas, por exemplo: a manutenção de regras de negócio requer que as mesmas sejam encontradas e modificadas no código de toda a aplicação; a regra de negócio só tem seu comportamento garantido se a implementação de toda a aplicação está correta; e a comunicação adicional entre a aplicação e o banco de dados é necessária depois de toda modificação² do banco de dados em questão. Uma segunda solução seria a criação de um processo de aplicação que periodicamente pesquisasse o banco de dados e verificasse se cada regra de negócio e cada gatilho foi disparado[WC96].

- “*roll back*” desnecessário de transações nos bancos de dados passivos, a validação das restrições de integridade sempre ocorre após cada operação que modifica² o banco de dados ou no final de uma transação. O resultado da violação destas restrições é o “*roll back*” da transação; o que seria evitado se pudéssemos, quando detectada a violação, realizar algumas ações que a retificasse.

As limitações mencionadas aqui dificultam o desenvolvimento de alguns tipos de aplicação, tais como: sistemas especialistas (um sistema especialista é um programa inteligente baseado no conhecimento de uma área de problema específica; este conhecimento é, basicamente, uma coleção de regras que permite ao sistema desenhar conclusões a partir de dados e premissas [For84] [Wat86]) que manipulam grandes quantidades de dados e o gerenciamento de fluxo de trabalho (tipicamente, monitoram o estado de certas tarefas e, quando apropriado, automaticamente invocam outras tarefas, passando os dados relevantes entre tais tarefas se necessário).

Nos anos 70, surgiram os primeiros projetos que defendiam um sistema de banco de dados que respondesse a eventos sem que o usuário ou a aplicação emitisse qualquer comando. Tais

² Inserção, atualização e exclusão.

bancos de dados foram chamados de **Bancos de Dados Ativos**. Os primeiros sistemas que apareceram foram: a linguagem de manipulação de dados CODASYL, criada pelo “Data Base Task Group” (DBTG); a “Query-by-Example” (QBE), que incluía os gatilhos (“triggers”) na verificação das restrições de integridade; e o subsistema de gatilhos presente no projeto de pesquisa do sistema de banco de dados relacional R (“System R”).

Mas só em meados dos anos 80 os bancos de dados ativos foram reconhecidos. Tal reconhecimento foi obtido com o surgimento de um projeto de banco de dados ativo pioneiro, o projeto HiPAC.

As figuras 1.1 e 1.2 ilustram a principal diferença entre os bancos de dados passivo e ativo. Na figura 1.2, observamos uma característica a mais, as regras ³. É justamente esta característica que diferencia esses dois tipos de bancos de dados.

A utilização de bancos de dados ativos facilita o desenvolvimento dos sistemas especialistas e dos gerenciadores de fluxo de trabalho mencionados anteriormente. No primeiro caso, as regras dos bancos de dados ativos são similares às regras da inteligência artificial utilizadas na elaboração desses sistemas. Quanto ao gerenciamento de fluxo de trabalho, o comportamento dos bancos de dados ativos (monitoramento e invocação de ações) e a execução de muitas tarefas sobre os dados fornecem uma forma conveniente e segura para este tipo de sistema. [WC96]

Além destes sistemas, os bancos de dados ativos podem ser utilizados na substituição de subsistemas pertencentes aos bancos de dados passivos, tais como: restrições de integridade simples, autorização, reunião de estatísticas e visões.

³ Discutiremos estas regras e estabeleceremos uma nomenclatura mais apropriada no próximo capítulo. Por enquanto, chamaremos de *regra* a característica que torna um banco de dado ativo.

Figura 1.1 - Exemplo do processamento em Bancos de Dados Passivos⁴.

⁴ Adaptado da página <http://www.cs.uga.edu/~sridarkp/active.html>.

Figura 1.2 - Exemplo do processamento em Bancos de Dados Ativos⁵.

1.2 Principais Problemas

Apesar de todas as vantagens dos bancos de dados ativos sobre os bancos de dados passivos, os bancos de dados ativos ainda não são largamente utilizados na prática. Dois fatos contribuem para esta situação:

- ❑ falta de uma metodologia de projeto de bancos de dados ativos que auxilie na determinação de qual parte do comportamento da aplicação será responsabilidade das regras dos bancos de dados ativos e como estas regras devem ser especificadas para fornecer, eficientemente, o comportamento desejado [WC96].
- ❑ dificuldade para prever o comportamento de um conjunto de regras; isto se deve principalmente à ausência do conceito de modularização ou estruturação na maioria das linguagens de regras existentes atualmente [WC96] [BW95].

⁵ Extraído de <http://www.cs.uga.edu/~sridarkp/active.html>.

Assim, a falta de estruturação, a dificuldade em prever o comportamento das regras e a ausência de uma metodologia de projeto de regras originam problemas, alguns deles descritos detalhadamente no próximo capítulo. Os principais problemas são os seguintes:

- ❑ terminação;
- ❑ confluência; e
- ❑ determinismo observável.

1.3 Utilização dos Bancos de Dados Ativos

Os sistemas de bancos de dados ativos são utilizados na implementação de várias aplicações. Segundo [WC96], tais aplicações podem ser divididas em três categorias:

- ❑ aplicações internas;
- ❑ aplicações estendidas; e
- ❑ aplicações externas.

Vejamos cada uma delas.

1.3.1 Aplicações Internas

Estas aplicações implementam algumas funcionalidades suportadas pelos sistemas de bancos de dados passivos através de regras, substituindo-as. Os principais exemplos desta categoria são:

- ❑ *gerenciamento de restrições de integridade* sabemos que as restrições de integridade fornecem meios para assegurar que mudanças feitas no banco de dados por usuários autorizados não resultem na perda de dados [KS93]; portanto, as regras intencionais que as implementam simplesmente verificam se tais restrições foram ou não violadas e, em caso afirmativo, executam ações corretivas que tratam a violação em questão; e
- ❑ *computação de visões*: as visões são relações lógicas baseadas em uma ou mais relações e/ou outras visões que podem ser gerenciadas de duas formas. A primeira corresponde às visões virtuais que não são armazenadas, mas todas as vezes que uma referência a uma destas visões é feita, seus dados são extraídos do banco de dados por uma consulta; tal extração pode ser feita por uma regra intencional. Também utilizamos as visões materializadas, cujos dados devem manter-se atualizados com suas relações base; para

tanto, qualquer atualização nestas tabelas devem ser propagadas para as visões que a utilizam, o que é facilmente implementado pelas regras intencionais.

1.3.2 Aplicações Estendidas

Estas aplicações utilizam as regras intencionais na implementação de funções que suportam tarefas de bancos de dados não padronizadas. Exemplos típicos são os sistemas que trabalham com grande quantidade de dados e cujas atividades possam ser divididas e processadas em servidores diferentes; neste caso, as regras intencionais podem coordenar os fluxos de dados e controle entre estes computadores. Outros exemplos desta categoria são os gerenciadores de versões e replicação.

1.3.3 Aplicações Externas

Nesta categoria, o comportamento reativo das regras intencionais é usado na implementação de sistemas pertencentes a um domínio específico. Alguns exemplos são controle de tráfego aéreo, controle de empréstimos de uma biblioteca, entre outros.

É importante ressaltar que no desenvolvimento desta dissertação nos concentraremos nesta categoria, fornecendo um exemplo que a ilustrará no capítulo 5.

1.4 Objetivos desta Dissertação

Esta dissertação tem como objetivo desenvolver um novo método para o projeto lógico de regras de bancos de dados ativos utilizando as tabelas de decisão, descrever o projeto de uma semântica confluyente para a execução destas regras, e apresentar um estudo de caso onde aplicamos nosso método de projeto de regras.

Para atingir tais objetivos, este trabalho foi dividido em seis capítulos.

Os primeiro e o último capítulos são a introdução e a conclusão, respectivamente.

No capítulo 2 são apresentados os principais conceitos de bancos de dados ativos e os problemas encontrados com sua utilização.

Em seguida, são discutidos dois métodos para resolução do problema da confluência de regras, entre os quais o projeto da semântica mencionada no início desta seção (capítulo 3).

Uma vez concluídos estes capítulos, serão discutidas as tabelas de decisão para bancos de dados ativos (capítulo 4), que constituem a ferramenta fundamental do nosso método de projeto de regras, e sua adaptação intuitiva à semântica confluyente discutida no capítulo 3.

Os conceitos e métodos consolidados no capítulo 4 serão utilizados no projeto de um banco de dados ativo para a Biblioteca “Carlos Benjamin de Lyra” do Instituto de Matemática e Estatística da USP, utilizando o modelo relacional.

Finalmente, discutiremos alguns assuntos básicos ao entendimento d esta dissertação nos apêndices A, B e C. Em primeiro lugar, apresentaremos as sintaxes das linguagens SQL (“Strutured Query Language”), PL/SQL (“Procedural Language / Structured Query Language”) e de regras de banco de dados ativos, a fim de mostrarmos as diferenças entre o SGBDR Oracle e a nossa abordagem para a linguagem de regras. Em seguida, descreveremos os conceitos e características fundamentais das tabelas de decisão. Enfim, detalharemos a implementação das Tabelas de Decisão para Banco de Dados Ativos apresentadas no estudo de caso (capítulo 5).

BANCOS DE DADOS ATIVOS

Neste capítulo apresentaremos o conceito e as principais características dos bancos de dados ativos e sua arquitetura.

2.1 O Que são os Bancos de Dados Ativos ?

São bancos de dados capazes de realizar automaticamente determinadas operações em resposta à ocorrência de certos eventos e/ou condições [WC96].

Os bancos de dados ativos possuem duas partes:

- *extensional* é a parte explícita do banco de dados. Esta parte é composta pelas estruturas internas⁶ do banco de dados, tais como tabelas, índices, visões, entre outras.
- *intencional* é a parte implícita do banco de dados, que é composta pelas regras reativas e pelas regras dedutivas⁷.

Nesta dissertação, nos concentraremos na parte intencional dos bancos de dados; mais precisamente nas regras reativas que, a partir deste momento, denominaremos **Regras Intencionais**.

2.2 Regras Intencionais

A principal característica dos bancos de dados ativos são as regras intencionais, pois tais regras determinam seu comportamento.

Tipicamente, as regras intencionais possuem três partes: evento, condição e ação. E podem ser escritas da seguinte forma:

"Quando EVENTO Se CONDIÇÃO Então AÇÃO"

Vamos discutir, agora, cada uma das partes de uma regra intencional.

⁶ Esta nomenclatura foi extraída de [Lon98].

⁷ São regras lógicas que concluem fatos a partir de uma ou mais premissas.

2.2.1 Evento

O **evento** especifica o que provocará o disparo de uma regra intencional.

Exemplo:

Seja uma regra intencional R com o evento “inserção de um exemplar na tabela Exemplar”.

Toda vez que um novo exemplar é cadastrado, a regra R será disparada.

Os tipos mais comuns de eventos são:

- ❑ *modificação do banco de dados*: este tipo de evento pode ser definido pelos comandos “insert”, “delete” ou “update”, da linguagem SQL, sobre uma tabela;
- ❑ *recuperação de dados*: o evento de recuperação é especificado pelo comando “select”, da linguagem SQL, sobre uma determinada tabela;
- ❑ *temporal*: estabelece que uma regra intencional será disparada em um tempo absoluto (15/dezembro/1997 às 15:00, por exemplo), em tempos repetidos (todo dia às 11:40, por exemplo) ou em intervalos periódicos (a cada 10 minutos, por exemplo); e
- ❑ *externos*: são eventos relacionados com o domínio da aplicação. Estes eventos são especificados pelos usuários ou administradores do banco de dados e recebem um nome E, por exemplo. Um exemplo de evento externo é o evento Empréstimo(VcodLivre, VnumDoc) num sistema bibliotecário.

Cabe ressaltar que podemos omitir os eventos. Neste caso, as regras sempre estarão sendo avaliadas.

2.2.2 Condição

A condição de uma regra intencional determina os testes adicionais a serem feitos quando uma destas regras é disparada.

Exemplo:

Seja uma regra intencional R com o evento “inserção de um exemplar na tabela Exemplar” e a condição “se existe o livro”. Suponhamos que o evento “inserção de um exemplar na tabela Exemplar” tenha ocorrido e a regra intencional R disparada; o próximo passo é avaliar a condição “se existe o livro que está sendo inserido”.

Alguns tipos de condição são:

- *predicados do banco de dados*: correspondem à cláusula “where” da linguagem SQL;
- *consultas ao banco de dados utilizando uma linguagem similar à SQL*: neste caso, devemos determinar se a condição é verdadeira quando a consulta produz uma resposta vazia ou se a condição é verdadeira quando a consulta produz uma resposta não vazia; e
- *procedimentos de aplicação*: estes procedimentos podem retornar valores “booleanos” ou dados; aqui o significado é semelhante ao caso anterior.

Assim como nos eventos, a condição de uma regra intencional pode ser omitida, em cujo caso assume-se que a condição é sempre verdadeira.

2.2.3 Ação

A ação de uma regra intencional é executada quando essa é disparada e sua condição é verdadeira.

Exemplo:

Seja o exemplo anterior. Suponhamos que o evento “inserção de um exemplar na tabela Exemplar” tenha ocorrido e a condição “se existe o livro” avaliada como verdadeira, então ação A, um “commit” por exemplo, será executada.

Os tipos mais comuns de ação são:

- *operações para modificação e recuperação de dados*: tais operações são representadas, por exemplo, pelos comandos “insert”, “delete”, “update” e “select” da linguagem SQL;
- *outros comandos de banco de dados*: como “rollback” e “commit”;
- *procedimentos de aplicação*: estes procedimentos são escritos pelos próprios usuários ou administradores de banco de dados e podem provocar um evento que dispare uma regra intencional.

É evidente que não faz o menor sentido omitir a ação de uma regra intencional.

2.2.4 Outras Características da Linguagem de Regras

2.2.4.1 Valores de Transição

Nos bancos de dados ativos, a maioria dos eventos ocorre quando o estado do banco de dados muda. E isto acontece devido a modificação de uma ou mais tuplas de um banco de dados relacional, por exemplo. Esta mudança de estado é conhecida como **transição** do banco de dados.

Diante destes fatos nós deparamos com a seguinte pergunta:

O que faremos se precisarmos, na avaliação da condição de uma regra intencional ou na execução de sua(s) ação(es), usar dados antigos ou dados que serão colocados no banco de dados após a execução desta regra?

Na verdade, as linguagens de regras usualmente incluem um mecanismo que faz referência aos valores (ou dados) associados a estas transições. Segundo [WC96], os mecanismos mais comuns são:

- ❑ *evento parametrizado*: o evento que dispara a regra intencional é parametrizado e estes parâmetros podem ser referenciados por suas condição e ação. Por exemplo, uma regra intencional pode ser disparada pelo evento “Reserva (VCodLivro, VNumDoc)”, onde VCodLivro e VNumDoc são dados a serem inseridos na tabela Reserva; neste caso, VCodLivro e VNumDoc poderão ser referenciados nas condição e ação desta regra. Em nosso estudo de caso utilizamos esta solução; e
- ❑ *tabelas de transição*: são tabelas lógicas que refletem as mudanças ocorridas durante a transição do banco de dados [AWH92]. Tais tabelas são acessadas através de palavras chaves como: “inserted”, “deleted”, “updated”, entre outras. Por exemplo, essas palavras quando usadas na condição ou na ação de uma regra intencional indicam os dados que foram inseridos, excluídos ou atualizados na transição do banco de dados em questão [WC96].

2.2.4.2 Prioridade entre Regras Intencionais

O que acontece se um evento dispara mais de uma regra intencional? Qual destas regras será avaliada primeiro? Como escolher?

As questões acima estão intimamente relacionadas à semântica de execução de regras intencionais, que será vista detalhadamente no item 2.3. O que nos interessa, aqui, é a necessidade de escolhermos uma regra intencional entre um conjunto de regras prontas para serem avaliadas. Com o propósito de controlar tais escolhas, algumas linguagens de regras intencionais permitem definir prioridades entre elas.

A prioridade entre regras intencionais pode ser de dois tipos: relativa e absoluta. No primeiro caso, devemos ordenar o conjunto de regras intencionais segundo a prioridade entre os pares de regras que o compõe. Já a prioridade absoluta é definida associando à regra um valor numérico que determina sua prioridade.

2.3 Semântica de Execução de Regras Intencionais

A semântica de execução de regras intencionais determina os possíveis comportamentos dos bancos de dados ativos, uma vez que um conjunto destas regras tenha sido definido [WC96].

Nesta seção, mostraremos o modelo fundamental da semântica de execução de regras intencionais e algumas características que complementam este modelo. Mas antes vamos entender a dependência entre as partes de uma regra intencional.

2.3.1 Dependência entre as Partes de uma Regra Intencional

As partes de uma regra intencional (evento, condição e ação) são dependentes entre si. Sendo que as duas principais dependências são:

- *Evento - Ação*: esta dependência é criada pela própria regra intencional, isto é, a ação de uma regra intencional r só será executada se o evento de r ocorrer e, consequentemente, fazê-la disparar.

$EVENTO \longrightarrow AÇÃO$

⁸ Quando falarmos em regras intencionais avaliadas, estamos considerando que a referida regra foi disparada e sua condição avaliada.

- *Ação - Evento*: tal dependência é gerada na execução da ação, que poderá gerar novos eventos no banco de dados (inserção de uma linha na tabela de empréstimos, por exemplo) e estes dispararão novas regras intencionais.

Analisando estas duas dependências, concluímos que existe um ciclo nos bancos de dados ativos (mostrado a seguir). Tal ciclo nos conduz a uma família de semânticas de execução que tratam as regras num ciclo conhecido como ECA (Evento – Condição – Ação).

2.3.2 Modelo Fundamental

No modelo fundamental da semântica de execução de regras intencionais trabalhamos, basicamente, com o ciclo ECA. Ou seja, descobrimos os eventos disparados, avaliamos as condições das regras cujos eventos ocorreram e, finalmente, executamos as ações necessárias.

O algoritmo que expressa este modelo é o seguinte [WC96]:

- (E) Detecte os eventos;
/* neste ponto, as regras que foram disparadas são descobertas */
 - (C) Avalie uma condição;
/* aqui, uma regra é escolhida para ser avaliada. Desta avaliação, a(s) ação(ões) é(são) gerada(s). */
 - (A) Execute, seletivamente, as ações;
/* a execução desta(s) ação(ões) pode(m) gerar um ou mais eventos. */
- Volte para E;

Neste algoritmo, observamos que nos passos (C) e (A) existe uma política de seleção, isto é, uma condição e uma ação são escolhidas dentro de um conjunto de condições e ações, nos passos (C) e (A), respectivamente.

2.3.3 Considerações a Respeito da Semântica de Execução de Regras Intencionais

O modelo fundamental, apresentado na seção anterior, pode ser complementado e/ou estendido a fim de aperfeiçoarmos a semântica de execução de regras.

Os principais pontos a serem considerados para atingirmos tal objetivo são [WC96]:

- ❑ granularidade do processamento de regras intencionais;
- ❑ execução orientada à instância *versus* orientada a conjunto;
- ❑ algoritmo iterativo *versus* recursivo;
- ❑ resolução de conflitos;
- ❑ execução sequencial *versus* concorrente;
- ❑ formas de acoplamento;
- ❑ terminação;
- ❑ confluência; e
- ❑ determinismo observável.

A seguir, detalharemos cada um destes pontos.

2.3.3.1 Granularidade do Processamento de Regras Intencionais

Durante o trabalho normal de um sistema de gerenciamento de banco de dados ativo, existem pontos ou instantes específicos nos quais as regras intencionais são avaliadas e suas ações executadas. A **granularidade do processamento de regras intencionais** determina a frequência desses pontos ou instantes [WC96].

Há vários níveis de granularidade. No mais fino, as regras intencionais são executadas em qualquer instante do trabalho do sistema se banco de dados, isto é, quando o evento ocorrer. No próximo nível de granularidade, as regras intencionais podem ser processadas após cada operação básica⁹ sobre o banco de dados. No nível mais grosso, as regras podem ser processadas ao término de cada comando que manipule dados¹⁰. Além destes três níveis principais, podemos ter a granularidade em nível de transações, o que não é usual [WC96].

⁹ Quando falamos em operação básica sobre o banco de dados nos referimos às operações “insert”, “update”, “delete” ou “select” sobre uma única tupla ou objeto.

¹⁰ Neste caso, consideramos a manipulação de mais de uma tupla ou objeto.

2.3.3.2 Execução Orientada à Instância *versus* Orientada a Conjunto

2.3.3.2.1 Execução Orientada à Instância

A execução de regras intencionais é **orientada à instância** quando uma regra é avaliada uma vez para cada instância do banco de dados ativo que a disparou ou satisfaz sua condição [WC96].

Exemplo:

Seja uma regra r_i que é disparada pelas exclusões em uma determinada tabela. Na execução orientada à instância, a condição de r_i é avaliada e sua ação executada uma vez para cada tupla excluída.

Segundo [WC96], devemos modificar o modelo fundamental da semântica de processamento de regras intencionais a fim de adaptá-lo à execução orientada à instância. Basicamente, devemos escolher entre:

- quando uma regra intencional é selecionada no passo (E), os passos (C) e (A) são executados para cada instância relevante à regra em questão; e
- uma vez que uma regra intencional é selecionada no passo (E), os passos (C) e (A) são executados para uma única instância relevante à regra; quando o laço do modelo fundamental continuar, outra regra intencional será selecionada podendo, inclusive, ser a mesma.

A primeira escolha é freqüentemente implementada como uma otimização do modelo fundamental da semântica de execução de regras intencionais.

2.3.3.2.2 Execução Orientada a Conjunto

Na **execução orientada a conjunto**, a regra intencional é avaliada uma vez para todas as instâncias do banco de dados que dispararam ou satisfizeram a sua condição [WC96].

Exemplo:

Seja uma regra r_i que é disparada pelas exclusões em uma determinada tabela. A condição de r_i é avaliada e sua ação executada uma vez para todo o conjunto de tuplas excluídas.

Segundo [WC96], o modelo fundamental da semântica de execução de regras intencionais não precisa ser alterado, pois os passos (C) e (A) são executados para todo o conjunto de instâncias relevantes à regra selecionada em (E).

2.3.3.3 Algoritmos Iterativos *versus* Algoritmos Recursivos

2.3.3.3.1 Algoritmos Iterativos

O modelo fundamental, apresentado na seção 2.3.2, é um **algoritmo iterativo**, isto é, uma regra intencional é selecionada e avaliada, depois uma segunda regra intencional passa por esse processo e assim sucessivamente.

No decorrer desta dissertação estaremos considerando apenas os algoritmos iterativos.

2.3.3.3.2 Algoritmos Recursivos

Seja um banco de dados ativo com três regras intencionais: r_i , r_j e r_k . Suponhamos que a regra intencional R_i esteja sendo avaliada, isto é, o algoritmo da semântica de processamento de regras está sendo executado para r_i ; consideremos, ainda, que o instante em que a semântica de processamento de regras deva ser executado novamente ocorra antes do término da avaliação de r_i . Neste caso, a execução mais recente deste algoritmo escolhe outra regra intencional para ser processada, ou seja, a semântica de execução de regras intencionais é invocada recursivamente [WC96]. Dizemos, então, que o **algoritmo** de processamento de regras é **recursivo**.

2.3.3.4 Resolução de Conflitos

Durante a atividade normal dos bancos de dados ativos, as regras intencionais são disparadas e precisam ser avaliadas. Em um dado momento, muitas regras intencionais podem estar disparadas e prontas para avaliação. Devemos, portanto, escolher apenas uma delas no passo (E) do modelo fundamental. Esta necessidade caracteriza a resolução de conflito entre regras intencionais [WC96].

Basicamente, devemos estabelecer uma política de seleção de regras; esta deve ser implementada no passo (E) do ciclo ECA. Segundo [WC96], existem várias:

- ❑ uma regra intencional pode ser escolhida arbitrariamente;
- ❑ uma regra intencional pode ser escolhida de acordo com a prioridade especificada na sua definição, conforme mostrado na seção 2.2.4.2;
- ❑ uma regra intencional pode ser escolhida de acordo com outras propriedades estáticas, tais como o instante de sua criação ou dos dados sobre os quais esta regra foi definida; e
- ❑ uma regra intencional pode ser escolhida de acordo com suas propriedades dinâmicas, tal como a mais recentemente disparada, por exemplo.

2.3.3.5 Execução Seqüencial *versus* Execução Concorrente

2.3.3.5.1 Execução Seqüencial

Até agora, só tratamos da execução seqüencial das regras intencionais, mesmo quando falamos a respeito dos algoritmos recursivos na seção 2.3.3.3.2. Basicamente, uma única regra intencional é processada por vez [WC96].

2.3.3.5.2 Execução Concorrente

Segundo [WC96], se múltiplas regras intencionais foram disparadas, podemos avaliar suas condições e executar suas ações simultaneamente. A este método de processamento de regras chamamos de **execução concorrente de regras intencionais**.

Por um lado, a execução concorrente evita o conflito entre regras intencionais e, como veremos mais tarde, auxilia na resolução do problema da confluência. Por outro, devemos adicionar à semântica de execução de regras intencionais mecanismos que impeçam a interferência do processamento de uma regra intencional na execução de outra.

2.3.3.6 Formas de Acoplamento

Estabelecer o relacionamento entre as regras intencionais e as transações de um banco de dados ativo consiste em determinar a **forma de acoplamento** entre elas, ou seja, dado uma regra intencional, quais partes de seu processamento serão feitas dentro de uma mesma transação e quais serão divididas em várias transações [WC96].

Segundo [WC96], as formas de acoplamento possíveis são:

- ❑ *imediata*: o evento ocorre, a condição é avaliada e a ação executada dentro de uma mesma transação;
- ❑ *adiada*: se o evento ocorre durante a transação T, então a avaliação da condição e a execução da ação da regra disparada são processadas no final desta transação; e
- ❑ *desacoplada*: neste caso, as partes da regra intencional são processadas em transações diferentes.

2.3.3.7 Terminação

Dizemos que a execução de um conjunto de regras intencionais R termina quando todas as execuções de R terminam para todos os possíveis estados do banco de dados, isto é, nenhuma regra intencional pertencente a R dispara outra regra ou a si mesma infinitamente.

Segundo [WC96], há várias maneiras de tratarmos a terminação de regras intencionais. As principais são:

- limitar o número de regras disparadas. Nesta abordagem, o processamento é suspenso quando este limite é atingido;
- construir o **grafo de disparos** e verificar a existência de ciclos; caso sejam encontrados, a terminação não é garantida. No grafo de disparos da figura 2.1, os nós correspondem a regras intencionais do conjunto de regras R e a presença de arcos de um nó r_i para um nó r_j indica que a execução da ação de r_i disparou a regra r_j .

Figura 2.1 - Grafo de disparo com ciclo [WC96].

Diante deste problema, podemos não admitir a formação de ciclos, suspendendo o processamento; ou permitir sua continuidade desde que asseguremos que a condição de uma destas regras torne-se falsa no decorrer do processamento; e

- a solução mais cômoda, deixar a cargo do desenvolvedor projetar regras intencionais que não acarretem o problema da terminação.

2.3.3.8 Confluência

Dizer se um conjunto R de regras intencionais é confluyente, consiste em responder a seguinte questão [WC96] [AWH92]:

A ordem de execução de regras intencionais sem prioridade entre si produz alguma diferença no estado final do banco de dados?

Se a resposta é não, podemos afirmar que o conjunto R é confluyente.

No próximo capítulo, detalharemos este problema e apresentaremos as principais soluções desenvolvidas para resolvê-lo.

2.3.3.9 Determinismo Observável

Um conjunto R de regras intencionais é deterministicamente observável se todas as ações visíveis produzidas pelo processamento de R são executadas na mesma ordem, independente da seqüência de execução de suas regras.

2.4 Arquitetura dos Sistemas de Bancos de Dados Ativos

Nesta seção, apresentaremos as três possíveis arquiteturas de bancos de dados ativos: em camadas, embutida e compilada.

2.4.1 Arquitetura em Camadas

Quando os componentes do banco de dados ativo estão localizados em uma camada acima do sistema de banco de dados passivo, conforme a figura 2.2, dizemos que a **arquitetura** deste banco de dados é **em camadas** [WC96].

Nesta arquitetura, o gerenciamento e processamento de regras intencionais é separado do sistema de banco de dados. Portanto, para monitorarmos os eventos que disparam as regras, devemos interceptar os comandos submetidos ao sistema de banco de dados ou os dados que são retornados ao usuário ou aplicação que os solicitou. Já as condições são avaliadas pelo processador de regras intencionais e as ações executadas através de chamadas a procedimentos de aplicação ou emissão de comandos ao banco de dados [WC96].

Neste estudo, adotaremos a arquitetura em camadas, apesar de estarmos utilizando o SGBDR Oracle 7.3. Tal escolha deve-se à utilização de uma semântica de execução de regras intencionais diferente daquela implementada no gerenciador de banco de dados em questão, conforme mostraremos no capítulo 3.

Figura 2.2 - Banco de Dados Ativo cuja arquitetura é em camadas.

Segundo [WC96], as vantagens de usarmos esta arquitetura são:

- ❑ os sistemas e bancos de dados passivos podem ser facilmente convertidos em sistemas ativos sem a modificação do núcleo do sistema de banco de dados;
- ❑ sistemas de bancos de dados passivos diferentes podem ser convertidos em sistemas de bancos de dados ativos que fornecem a mesma “interface”.

Já as desvantagens são:

- ❑ baixo desempenho devido à intensa comunicação entre a camada ativa e o banco de dados passivo; e
- ❑ como a camada ativa não pode interagir com subsistemas do banco de dados passivo ¹¹, algumas características que requerem acesso a estes subsistemas não podem ser implementadas ¹².

2.4.2 Arquitetura Embutida

Na **arquitetura embutida**, todos os componentes do banco de dados ativo fazem parte do sistema de banco de dados, isto é, todo o gerenciamento e processamento de regras intencionais está integrado ao SGBD (Sistema de Gerenciamento de Bancos de Dados) [WC96]. A figura 2.3 mostra um sistema de banco de dados ativo com esta arquitetura.

¹¹ Tais como formas de acoplamento, controle de concorrência , etc.

Figura 2.3 - Banco de Dados Ativo cuja arquitetura é embutida, adaptado de [KA93].

As vantagens desta arquitetura são:

- ❑ bom desempenho no processamento de regras, visto que este é feito dentro do SGBD; e
- ❑ acesso aos subsistemas do banco de dados que permitem a implementação de características mais avançadas.

As desvantagens são:

- ❑ complexidade para implementação e manutenção do sistema de banco de dados; e
- ❑ heterogeneidade dos sistemas de bancos de dados ativos.

2.4.3 Arquitetura Compilada

A **arquitetura compilada** não requer nenhuma atividade em tempo de execução; pois quando os procedimentos de aplicação ou as operações sobre o banco de dados são compiladas, eles são modificados para incluir os efeitos das regras intencionais relacionadas [WC96].

Suas vantagens são [WC96]:

- ❑ o monitorização dos eventos e avaliação das condições das regras intencionais não são necessários;
- ❑ redução da complexidade da tarefa de implementação do banco de dados; e
- ❑ aumento do desempenho do sistema.

¹² Tais como, gerenciamento de transações e bloqueio, componente de autorização, etc.

Por outro lado, a desvantagem é a seguinte:

- uso de linguagens de aplicação e de regras intencionais restritas. Um exemplo desta limitação é a necessidade de detectarmos todos os eventos em tempo de compilação, assim, não podemos implementar os eventos temporais.

2.5 Comparação entre o POSTGRES, o Ariel, o Starburst e a Extensão “Ativa” para Bancos de Dados Relacionais desta Dissertação.

A tabela a seguir resume as principais características dos projetos de bancos de dados ativos POSTGRES, Ariel, Starburst e a extensão “ativa” que desenvolveremos no restante desta dissertação.

	<i>POSTGRES</i>	<i>Ariel</i>	<i>Starburst</i>	<i>Extensão Ativa</i>
<i>Granularidade do processamento de regras</i>	Toda vez que um comando POSTQUEL é emitido	Após cada transação do banco de dados	Em geral, após cada transição do banco de dados. Também pode ocorrer devido a execução dos comandos <i>process rule</i> , <i>process ruleset</i> .	Fina
<i>Execução orientada à instância ou orientada a conjunto</i>	Orientada a conjunto	Orientada a conjunto	Orientada a conjunto	Orientada a conjunto
<i>Execução Sequencial ou Concorrente</i>	Execução sequencial	Execução sequencial	Execução sequencial	Execução concorrente
<i>Algoritmo iterativo ou recursivo</i>	Iterativo	Iterativo	Iterativo	Iterativo

	<i>POSTGRES</i>	<i>Ariel</i>	<i>Starburst</i>	<i>Extensão Ativa</i>
<i>Resolução de conflitos</i>		Geralmente, com prioridades entre regras.	Prioridade entre regras.	Consequência da execução concorrente.
<i>Valores de Transição</i>	Tabelas de transição	Tabelas de transição	Tabelas de transição	Eventos parametrizados
<i>Forma de acoplamento</i>	Imediata	Adiada.	Adiada, a menos dos comandos <i>process rule</i> , <i>process ruleset</i> .	Imediata
<i>Terminação</i>			Análise estática de regras (implementada em uma extensão).	
<i>Confluência</i>		A princípio seria garantida pelas prioridades entre as regras.	Análise estática de regras.	Semântica de execução de regras intencionais.
<i>Arquitetura</i>	Embutida	Embutida	Em camadas	Em camadas
<i>Outras Características</i>	Regras intencionais com encaadeamento “backward” e “forward”	Rede de discriminação cujo objetivo é otimizar a avaliação das condições das regras.		

CONFLUÊNCIA DE REGRAS INTENCIONAIS

Uma das maiores desvantagens dos sistemas de bancos de dados ativos é a dificuldade de predizermos como um conjunto de regras intencionais se comportará durante o seu processamento. Existem vários problemas decorrentes desta desvantagem, entre eles podemos citar a confluência de regras intencionais. Tal problema é seríssimo, pois pode gerar bancos de dados inconsistentes e, conseqüentemente, anular os esforços para desenvolvimento e manutenção dos mesmos.

Neste capítulo, descreveremos este problema e os dois tipos principais de métodos para resolvê-lo, entre os quais o nosso projeto de semântica confluyente.

3.1 O Que é um Conjunto de Regras Intencionais Confluentes?

De acordo com a seção 2.3.3.8, um **conjunto de regras intencionais** é **confluyente** se o estado final do banco de dados, após o término do processamento destas, não depende da regra escolhida para avaliação quando várias outras estavam disparadas [AWH92].

Cabe ressaltar que só devemos nos preocupar com as regras intencionais que não possuem prioridades entre si e estão disparadas simultaneamente, pois caso existam prioridades ou apenas uma regra intencional esteja disparada a cada ciclo ECA a ordem de processamento das regras já está definida e, conseqüentemente, o estado final do banco de dados será único.

3.2 Soluções para o Problema da Confluência

Apresentaremos aqui duas maneiras de resolver o problema da confluência de regras intencionais. Na primeira solução, a semântica de execução de regras é mantida e fazemos a análise estática sobre tal propriedade. Já na segunda solução, modificamos a semântica.

3.2.1 Análise Estática

Neste item, descreveremos um método de resolução do problema da confluência baseado na análise estática de um conjunto de regras intencionais. Este método foi desenvolvido por Alexander Aiken, Jennifer Widom e Joseph M. Hellerstein no IBM Almaden Research Center e é baseado num protótipo do sistema gerenciador de banco de dados Starburst.

Basicamente, o método em questão é um algoritmo conservador¹³ que recebe um conjunto de regras intencionais e garante sua confluência ou diz que este conjunto pode não ser confluentes. Quando a resposta é a última, o algoritmo em questão isola as regras intencionais responsáveis pelo problema e determina critérios que garantem tal propriedade [AWH92].

Começaremos o estudo deste método apresentando os grafos de execução de regras e a definição de regras comutativas. Em seguida veremos o método de análise mencionado e faremos considerações a respeito de sua utilização.

3.2.1.1 Grafo de Execução de Regras Intencionais

3.2.1.1.1 Definição

Um **grafo de execução de regras intencionais** é um grafo orientado cujos vértices representam estados do processamento de regras e os arcos, as regras escolhidas para avaliação [AWH92], conforme apresentado na figura 3.1.

Figura 3.1 - Exemplo de um grafo de execução de regras intencionais.

Neste grafo, cada vértice ou estado S do processamento de regras tem dois componentes:

- um estado do banco de dados, representado por D ; e
- um conjunto TR de regras disparadas e suas tabelas de transição associadas.

E é representado por $S = (D, TR)$.

¹³ Entenderemos melhor esta afirmação nas próximas seções.

Além disso, um caminho de S_1 a S_2 de tamanho zero ou mais é denotado por:

$$S_1 \xrightarrow{*} S_2$$

3.2.1.1.2 Propriedades

Consideremos um conjunto de regras intencionais R e um arco num grafo de execução do estado $S_1 = (D_1, TR_1)$ para o estado $S_2 = (D_2, TR_2)$ representando a escolha da regra r para processamento. Segundo [AWH92], este grafo possui as seguintes propriedades:

- r pertence ao conjunto de regras elegíveis à avaliação, isto é, r foi disparada e não existe nenhuma outra regra r' disparada com prioridade superior a r .
- Além disso,
 - 1) as ações de r provocam um “rollback” da transação; ou
 - 2) existe um subconjunto O das operações executadas pela ação de r tal que as regras disparadas em TR_2 podem ser derivadas das regras disparadas em TR_1 :
 - (a) removendo-se r ;
 - (b) removendo-se todas as regras cujas mudanças que as dispararam foram desfeitas pelas operações em O ; e
 - (c) adicionando-se todas as regras $r' \in R$ tal que as ações de r as dispare.

3.2.1.2 Comutatividade entre Regras

Seja um conjunto R de regras intencionais. Dadas duas regras r_i e r_j pertencentes a R , ambas sem prioridades e cujas ações não dispararam nenhuma outra regra, e lembrando o conceito de confluência, seção 3.1, dizemos que estas regras são confluente se a partir de um estado S do grafo de execução, considerando r_i e então r_j produzirá o mesmo estado final S' se considerássemos r_j e, em seguida, r_i . Este é, justamente, o conceito de regras comutativas, figura 3.2 [AWH92].

A seguir apresentaremos condições que, se satisfeitas, nos asseguram que duas regras não comutam. Mas antes, vejamos a definição de alguns conjuntos¹⁴:

- $R = \{r_1, r_2, \dots, r_n\}$ é o conjunto das regras que serão analisadas;

¹⁴ Tais definições serão utilizadas no restante deste capítulo.

- $P = \{r_i > r_j, r_k > r_l, \dots\}$ estabelece as prioridades entre as regras definidas pelo usuário, onde $r_i > r_j$ indica que r_i tem precedência sobre r_j ;
- $T = \{t_1, t_2, \dots, t_m\}$ é o conjunto de tabelas do banco de dados;
- $O = \{<I, t> \mid t \in T\} \cup \{<D, t> \mid t \in T\} \cup \{<U, t, c> \mid t, c \in C\}$ é o conjunto de operações que modificam o banco de dados; onde $<I, t>$ são as inserções na tabela t , $<D, t>$ exclusões da tabela t e $<U, t, c>$ é a atualização da coluna c da tabela t .

Segundo [AWH92], o lema abaixo é utilizado para verificarmos se duas regras comutam ou não.

Figura 3.2 - Regras comutativas.

Lema 3.1: Sejam duas regras distintas r_i e r_j e o conjunto O de operações. Se qualquer uma das condições abaixo são satisfeitas então r_i e r_j podem ser não comutativas; caso contrário elas comutam:

- (a) a ação de r_i dispara r_j ;
- (b) a ação de r_i desfaz as mudanças que dispararam r_j ;
- (c) a ação de r_i altera informações nas colunas utilizadas na avaliação da condição e execução da ação de r_j ;
- (d) as inserções da ação de r_i em uma tabela t podem afetar o que a ação de r_j atualiza ou deleta na mesma tabela;
- (e) ambas as ações de r_i e r_j atualizam a mesma coluna na tabela t .

Analisando cuidadosamente este lema, observamos que o mesmo é muito restritivo. Imaginemos que as regras r_i e r_j atualizem a mesma coluna na tabela t , mas nunca na mesma tupla.

Assim, pelo lema anterior estas regras não comutam, mas sabemos que a atualização feita por r_i não influi na feita por r_j e, portanto, r_i e r_j são comutativas.

Vejamos, informalmente, o método para análise da confluência e como resolver esse inconveniente. Para uma abordagem formal consulte [AWH92].

3.2.1.3 Método para Análise da Confluência

Na figura 3.2, mostramos um par de regras intencionais confluentes. Entretanto esta não é a situação mais freqüente (lembre-se que ambas as regras não dispararam nenhuma outra), na prática cada uma destas regras pode disparar outras, o que torna a análise do conjunto R mais complexa.

Assim, devemos provar que a partir de um estado S de um grafo de execução GE , a escolha de qualquer uma das regras elegíveis, r_i ou r_j , produz o mesmo estado final S' neste grafo, conforme apresentado na figura 3.3.

Figura 3.3 - Confluência de regras intencionais.

Cabe ressaltar na figura 3.3 que no caminho entre S_i e S' estamos considerando as regras intencionais r_i e todas as regras intencionais que por ventura sejam disparadas por r_i e r_j . Situação análoga ocorre no caminho entre S_j e S' .

Como r_i e r_j comutam, r_j pode ser avaliada a partir do estado S_j . Entretanto, o que acontece se r_i disparar uma regra r que possua prioridade maior que r_j ? Nesse caso, r deve ser processada antes de r_j e ambas devem ser comutativas. Análise análoga é feita a partir do estado S_j .

Portanto, estamos construindo um grafo com dois caminhos p_1 e p_2 a partir dos estados S_i e S_j respectivamente. Neste grafo de execução, as regras intencionais com prioridade maior que r_j formam o conjunto R_1 , sendo executadas a partir de S_i ; em seguida podemos executar r_j , pois não há regras com prioridade maior que esta, o que origina o estado S' . Similarmente, as regras intencionais com precedência maior que r_i compõem o conjunto R_2 e são executadas a partir de S_j ; executamos, então, r_i e chegamos a S' (figura 3.4).

Figura 3.4 - Grafo de execução representando parte dos caminhos p_1 e p_2 .

Continuando o caminho p_1 , podemos, a partir de S'_i processar as regras intencionais no conjunto R_2 , que devem comutar com r_i e todas as regras pertencentes ao conjunto R_j . Analogamente, executamos as regras em R_i a partir do estado S'_j . Obtendo o estado final S' e, conseqüentemente, provando a confluência das regras intencionais em r_i e r_j (figura 3.5).

Figura 3.5 - Grafo de execução cujos caminhos p_1 e p_2 convergem para o mesmo estado final.

Segundo [AWH92], a situação descrita acima não é necessariamente válida, pois não sabemos se há outras regras intencionais com prioridade maior que as regras do conjunto R_2 no estado S'_i (por exemplo, uma das regras intencionais pertencentes a R_2 podem não ser elegíveis no estado S'_i pois alguma regra em R_i provocou um evento que disparou uma regra com prioridade menor que r_j e superior a regra pertencente a R_2 , mencionado anteriormente); o mesmo problema ocorre com S'_j com o conjunto R_j . Para evitar esta situação, os autores propuseram que o conjunto R_i fosse composto de

todas as regras com prioridade superior às regras do conjunto R_2 e vice-versa. Assim, se as regras intencionais $\{r_i\} \cup R_1$ comutam com $\{r_j\} \cup R_2$, então garantimos a existência do estado S' e, consequentemente, a confluência. Tais considerações originam os seguintes requisitos para confluência de duas regras, extraído de [AWH92]:

Definição: Considere duas regras intencionais r_i e r_j , sem prioridade entre si, pertencentes ao conjunto R . Seja $R_1 \subseteq R$ e $R_2 \subseteq R$. Construído pelo seguinte algoritmo:

$$R_1 \leftarrow \{r_i\}$$

$$R_2 \leftarrow \{r_j\}$$

Repita até que os conjuntos R_1 ou R_2 não mudem

$$R_1 \leftarrow R_1 \cup \{r \text{ é disparada por } r_i \text{ para alguma regra } r_1 \in R_1 \text{ e } r > r_2 \in P \text{ para alguma regra } r_2 \in R_2 \text{ e } r \neq r_j\}$$

$$R_2 \leftarrow R_2 \cup \{r \text{ é disparada por } r_2 \text{ para alguma regra } r_2 \in R_2 \text{ e } r > r_1 \in P \text{ para alguma regra } r_1 \in R_1 \text{ e } r \neq r_j\}$$

Para todo par de regras $r_1 \in R_1$ e $r_2 \in R_2$, r_1 e r_2 devem comutar.

Assim, o método apresentado em [AWH92] pode ser resumido da seguinte forma:

Se não há caminhos infinitos em nenhum grafo de execução para o conjunto R Então

Para cada par de regras r_i e r_j em R

Construa os conjuntos R_1 e R_2

Se as regras em R_1 comutam com as regras de R_2 então o conjunto R é conflúente.

3.2.1.4 Considerações

Antes de encerrarmos a discussão do método fornecido em [AWH92], devemos fazer duas considerações:

- em primeiro lugar, quando a análise do conjunto R determina que o mesmo não é conflúente, é fácil isolar as regras intencionais que violaram esta propriedade. Frequentemente, estas regras (r_i e r_j , por exemplo) são aquelas utilizadas para a construção

dos conjuntos R_1 e R_2 e não comutam; identificadas as regras r_i e r_j , a melhor solução é definirmos uma prioridade entre elas; e

- finalmente, o método aqui apresentado baseia-se no lema 3.1 e, como vimos anteriormente, o mesmo é restritivo.

3.2.2 Modificação da Semântica de Execução de Regras Intencionais

Nesta seção apresentaremos o projeto de uma semântica de execução de regras intencionais que resolve o problema da confluência no decorrer do seu processamento. Tal semântica baseia-se na semântica de execução descrita no projeto TEMPORA¹⁵ [BNP91], além de contemplar as principais características presentes em qualquer semântica de regras.

Antes de abordarmos essa semântica, vamos definir e exemplificar os conjuntos de ações consistentes.

3.2.2.1 Conjunto de Ações Consistentes

Consideremos duas regras intencionais r_i e r_j , e a tabela $T(\underline{X}, Y, Z)$ pertencente a um determinado banco de dados. Seja também as ações a_i e a_j correspondente às regras r_i e r_j , respectivamente. Dizemos que estas ações não são consistentes se uma das seguintes condições são verdadeiras:

- $a_i = \text{INSERT INTO } T \text{ VALUES}(x, y, z) \text{ e } a_j = \text{DELETE FROM } T \text{ WHERE } X=x;$
- $a_i = \text{UPDATE } T \text{ SET } Y = y' \text{ WHERE } X=x \text{ e } a_j = \text{UPDATE } T \text{ SET } Y = y'' \text{ WHERE } X=x;$ e
- $a_i = \text{UPDATE } T \text{ SET } Y = y' \text{ WHERE } X=x \text{ e } a_j = \text{DELETE FROM } T \text{ WHERE } X=x.$

Dizemos que um conjunto de ações $\mathcal{A} = \{a_1, a_2, \dots, a_n\}$ é consistente se não existirem duas ações a_i e $a_j \in \mathcal{A}$ conflitantes.

¹⁵ O projeto TEMPORA foi desenvolvido com a colaboração do Imperial College, Inglaterra; BIM, Bélgica; Hitec, Grécia; LPA, Inglaterra; SINTEF, Noruega; SISU, Suécia; “University of Liège, Bélgica; e UMIST, Inglaterra. O principal objetivo deste projeto é utilizar a modelagem de regras de negócio associada ao processo de desenvolvimento de software; para tanto, três modelos conceituais são utilizados: o modelo de Entidade-

Exemplo:

*O conjunto de ações $A' = \{UPDATE\ Leitor\ SET\ Classificacao = P\ WHERE\ CodLeitor = 101, UPDATE\ Leitor\ SET\ DtInscricao = 01/03/98\ WHERE\ CodLeitor = 101, UPDATE\ Leitor\ SET\ DtInscricao = 02/03/98\ WHERE\ CodLeitor = 102\}$ é consistente. Já o conjunto $A'' = \{UPDATE\ Leitor\ SET\ DtInscricao = 01/03/98\ WHERE\ CodLeitor = 101, UPDATE\ Leitor\ SET\ DtInscricao = 02/03/98\ WHERE\ CodLeitor = 101\}$ não é consistente, pois duas atualizações são feitas sobre as mesmas linha e coluna da tabela *Leitor*.*

3.2.2.2 Semântica Confluente

A origem do problema da confluência de regras intencionais é a necessidade de escolhermos uma regra, dentre várias regras intencionais disparadas, para avaliação.

Apresentaremos, agora, o algoritmo iterativo que retrata o modelo de execução confluyente de regras intencionais e é executado toda vez que uma ou mais regras estão disparadas.

A idéia básica deste algoritmo é executar concorrentemente todas as regras do banco de dados ativo que estejam disparadas em um determinado instante, isto é, avaliar simultaneamente todas as condições destas regras para todas as instâncias do banco de dados que a dispararam e, em seguida, executar suas ações.

O algoritmo que expressa esta semântica é o seguinte:

(E) Construa o conjunto E de todos os eventos detectados;

(C) Avalie as condições de todas as regras cujos eventos pertençam a E. A partir desta avaliação, construa o conjunto C de todas as condições verdadeiras;

Apague o conjunto E;

(A) Gere as ações a serem executadas, colocando-as no conjunto A;

Se A é Consistente

Então Execute todas as ações em A;

Senão Aborto;

Volte para o passo E;

Analisando este algoritmo, observamos duas características extremamente importantes que fazem desta semântica um modelo de execução confluyente de regras intencionais.

A primeira delas é a execução concorrente de todas as regras intencionais que estavam disparadas quando este algoritmo iniciou sua execução. Este tipo de execução evita a escolha de uma destas regras para avaliação e, conseqüentemente, a causa do problema da confluência.

Já a segunda característica é a verificação da consistência das ações das regras cujas condições foram avaliadas como verdadeiras. Após esta verificação, temos a certeza de que a execução destas ações não são conflitantes e, portanto, sua ordem de execução não interfere no estado final do banco de dados.

Assim, o algoritmo acima implementa uma semântica de execução de regras intencionais confluyente.

No próximo capítulo, uniremos esta semântica a um novo método de desenvolvimento de regras intencionais.

Capítulo 4

TABELAS DE DECISÃO COMO FORMA DE REPRESENTAR REGRAS INTENCIONAIS

Neste capítulo apresentaremos as **Tabelas de Decisão para Bancos de Dados Ativos**. Tais tabelas são uma extensão das tabelas de decisão amplamente difundidas na literatura. Uma discussão detalhada sobre estas tabelas encontra-se no Apêndice B.

4.1 Tabelas de Decisão para Bancos de Dados Ativos

Nos capítulos anteriores, constatamos que elaborar sistemas de informação baseados em regras intencionais não é uma tarefa simples. Parte desta dificuldade deve-se a ausência de uma ferramenta que nos permita analisar a interação entre as regras que possuem o mesmo evento ou que suas ações disparem outras regras.

Pensando nesta dificuldade e analisando as vantagens trazidas pelas tabelas de decisão à programação estruturada, resolvemos desenvolver as **Tabelas de Decisão para Bancos de Dados Ativos**.

A seguir apresentaremos esta nova ferramenta, suas características, bem como sua adequação à linguagem apresentada no Apêndice A.

4.1.1 O Que é uma Tabela de Decisão para Banco de Dados Ativos?

Uma **Tabela de Decisão para Banco de Dados Ativos (TDBDA)** é uma representação tabular dos eventos, condições e ações das regras intencionais.

Convencionamos uma estrutura para representação das TDBDA's, que é apresentada abaixo:

<i>On Reserva(VcodLivro, VNumDoc)</i>			
<i>EstáDisponível(VcodLivro)</i>	<i>V</i>	<i>F</i>	<i>F</i>
<i>"Verifica se o livro VcodLivro está com o leitor VnumDoc"</i>	<i>-</i>	<i>V</i>	<i>F</i>
<i>"Reserva do livro"</i>			<i>1</i>
<i>msg('A reserva não pode ser feita, pois o livro está disponível.')</i>	<i>1</i>		
<i>msg('A reserva não pode ser feita, pois o leitor está com o livro.')</i>		<i>1</i>	
<i>Commit</i>			<i>2</i>
<i>Rollback</i>	<i>2</i>	<i>2</i>	

Figura 4.1 - Estrutura de representação de uma TDBDA.

4.1.2 Interpretação e Estrutura de uma TDBDA

Consideremos a TDBDA apresentada na figura 4.1, esta pode ser interpretada da seguinte maneira:

“Caso seja solicitada a reserva de um livro, avalie as condições a seguir e execute as ações apropriadas:

- ❑ se o livro *VcodLivro* está disponível, então apresente a mensagem ‘A reserva não pode ser feita pois o livro está disponível’ e desfaza a transação;
- ❑ se o livro *VcodLivro* não está disponível e está emprestado ao leitor *VNumDoc*, então apresente a mensagem ‘A reserva não pode ser feita pois o leitor está com o livro’ e desfaza a transação; e
- ❑ se o livro não está disponível e não está emprestado ao leitor *VNumDoc*, então reserve o livro e confirme a transação.”

Considerando a interpretação acima e a TDBDA da figura 4.1, podemos observar as seguintes características:

- ❑ As TDBDA's são compostas por três partes: **linha de evento**, **linhas de condição** e **linhas de ação** conforme apresentado nas figuras 4.2, 4.3 e 4.4, respectivamente.

<i>On Reserva(VCodLivro, VNumDoc)</i>			
<i>EstáDisponível(VCodLivro)</i>	V	F	F
<i>"Verifica se o livro VCodLivro está com o leitor VNumDoc"</i>	-	V	F
<i>"Reserva do livro"</i>			1
<i>msg('A reserva não pode ser feita, pois o livro está disponível.')</i>	1		
<i>msg('A reserva não pode ser feita, pois o leitor está com o livro.')</i>		1	
<i>Commit</i>			2
<i>Rollback</i>	2	2	

Figura 4.2 - Linha de evento de uma TDBDA.

<i>On Reserva(VCodLivro, VNumDoc)</i>			
<i>EstáDisponível(VCodLivro)</i>	V	F	F
<i>"Verifica se o livro VcodLivro está com o leitor VNumDoc"</i>	-	V	F
<i>"Reserva do livro"</i>			1
<i>msg('A reserva não pode ser feita, pois o livro está disponível.')</i>	1		
<i>msg('A reserva não pode ser feita, pois o leitor está com o livro.')</i>		1	
<i>Commit</i>			2
<i>Rollback</i>	2	2	

Figura 4.3 - Linhas de condição de uma TDBDA.

<i>On Reserva(VCodLivro, VNumDoc)</i>			
<i>EstáDisponível(VCodLivro)</i>	V	F	F
<i>"Verifica se o livro VcodLivro está com o leitor VNumDoc"</i>	-	V	F
<i>"Reserva do livro"</i>			1
<i>msg('A reserva não pode ser feita, pois o livro está disponível.')</i>	1		
<i>msg('A reserva não pode ser feita, pois o leitor está com o livro.')</i>		1	
<i>Commit</i>			2
<i>Rollback</i>	2	2	

Figura 4.4 - Linhas de ação de uma TDBDA.

- As linhas de condição possuem duas seções: **descrição das condições** (figura 4.5) e **entradas de condições** (figura 4.6).

<i>On Reserva(VCodLivro, VNumDoc)</i>			
<i>EstáDisponível(VCodLivro)</i>	V	F	F
<i>"Verifica se o livro VcodLivro está com o leitor VNumDoc"</i>	-	V	F
<i>"Reserva do livro"</i>			1
<i>msg('A reserva não pode ser feita, pois o livro está disponível.')</i>	1		
<i>msg('A reserva não pode ser feita, pois o leitor está com o livro.')</i>		1	
<i>Commit</i>			2
<i>Rollback</i>	2	2	

Figura 4.5 - Descrição das condições de uma TDBDA.

On Reserva(VCodLivro, VNumDoc)			
EstáDisponível(VCodLivro)	V	F	F
"Verifica se o livro VcodLivro está com o leitor VNumDoc"	-	V	F
"Reserva do livro"			1
msg('A reserva não pode ser feita, pois o livro está disponível.')	1		
msg('A reserva não pode ser feita, pois o leitor está com o livro.')		1	
Commit			2
Rollback	2	2	

Figura 4.6 - Entradas de condições de uma TDBDA.

- As linhas de ação são divididas em **descrição das ações** e **entradas de ações**, figuras 4.7 e 4.8 respectivamente.

On Reserva(VCodLivro, VNumDoc)			
EstáDisponível(VCodLivro)	V	F	F
"Verifica se o livro VcodLivro está com o leitor VNumDoc"	-	V	F
"Reserva do livro"			1
msg('A reserva não pode ser feita, pois o livro está disponível.')	1		
msg('A reserva não pode ser feita, pois o leitor está com o livro.')		1	
Commit			2
Rollback	2	2	

Figura 4.7 - Descrição das ações de uma TDBDA.

On Reserva(VCodLivro, VNumDoc)			
EstáDisponível(VCodLivro)	V	F	F
"Verifica se o livro VcodLivro está com o leitor VNumDoc"	-	V	F
"Reserva do livro"			1
msg('A reserva não pode ser feita, pois o livro está disponível.')	1		
msg('A reserva não pode ser feita, pois o leitor está com o livro.')		1	
Commit			2
Rollback	2	2	

Figura 4.8 - Entradas de ação de uma TDBDA.

Note que as entradas de ação contém números que indicam a ordem de sua execução na regra a qual faz parte.

- A i-ésima entrada de condição de todas as linhas de condição formam uma **coluna de entradas de condição**, figura 4.9. Analogamente, a i-ésima entrada de ação de todas as linhas de ação formam uma **coluna de entradas de ação**, figura 4.10.

On Reserva(VCodLivro, VNumDoc)			
EstáDisponível(VCodLivro)	V	F	F
"Verifica se o livro VcodLivro está com o leitor VNumDoc"	-	V	F
"Reserva do livro"			1
msg('A reserva não pode ser feita, pois o livro está disponível.')	1		
msg('A reserva não pode ser feita, pois o leitor está com o livro.')		1	
Commit			2
Rollback	2	2	

Figura 4.9 - Coluna de entradas de condição de uma TDBDA.

On Reserva(VCodLivro, VNumDoc)			
EstáDisponível(VCodLivro)	V	F	F
"Verifica se o livro VcodLivro está com o leitor VNumDoc"	-	V	F
"Reserva do livro"			1
msg('A reserva não pode ser feita, pois o livro está disponível.')	1		
msg('A reserva não pode ser feita, pois o leitor está com o livro.')		1	
Commit			2
Rollback	2	2	

Figura 4.10 - Coluna de entradas de ação de uma TDBDA.

Podemos notar que cada coluna de entradas de condição corresponde a uma única coluna de entradas de ação e vice-versa, formando, portanto, uma regra situação-ação¹⁶.

Exemplo:

As colunas de entradas de condição e ação das figuras 4.9 e 4.10 formam a seguinte regra situação-ação:

```
WHEN Reserva(VCodLivro, VNumDoc)
  IF Not (EstáDisponível(VCodLivro)) AND "O livro está com o Leitor" THEN
    msg('A reserva não pode ser feita, pois o leitor não está com o livro. ');
    Rollback;
```

4.1.3 Outras Características

- Tabelas de decisão para bancos de dados ativos podem ser classificadas em limitada (figura 4.11), estendida (figura 4.12) e mista (figura 4.13). Nas **TDBDA's limitadas**, as entradas de condições assumem dois valores: V (verdadeiro) ou F (falso). Já nas **TDBDA's estendidas**, as descrições das condições são expressões que devem ser complementadas com o conteúdo das respectivas entradas de condição para se tornar uma expressão lógica

¹⁶ Segundo [Tan95], são regras em que uma situação ocorre quando um evento é detectado e uma condição é satisfeita.

[Nag90]. Finalmente, as **TDBDA's mistas** possuem parte das descrições de condições similares às limitadas e parte como das tabelas estendidas.

On DevolveLivroIME(VCodLivro, VNumDoc)			
$DtDevoluçãoLivroIME(VcodLivro, VNumDoc) \leq Sysdate$	V	F	F
$NumAtrasosIME(VNumDoc) < 3$		V	F
"Registra a Devolução do livro."	1	1	1
"Suspende o leitor."			2
"Atualiza o número de atrasos do leitor."		2	3
Commit	2	3	4

Figura 4.11 - Exemplo de uma TDBDA limitada.

On DevolveLivroIME(VCodLivro, VNumDoc)			
$DtDevoluçãoLivroIME(VcodLivro, VnumDoc)$	$\leq Sysdate$	$> Sysdate$	$\geq Sysdate$
$NumAtrasosIME(VNumDoc)$	-	< 3	≥ 3
"Registra a Devolução do livro."	1	1	1
"Suspende o leitor."			2
"Atualiza o número de atrasos do leitor."		2	3
Commit	2	3	4

Figura 4.12 - Exemplo de TDBDA estendida.

On DevolveLivroIME(VCodLivro, VNumDoc)			
$DtDevoluçãoLivroIME(VcodLivro, VNumDoc) = < Sysdate$	V	F	F
$NumAtrasosIME(VNumDoc)$	-	< 3	≥ 3
"Registra a Devolução do livro."	1	1	1
"Suspende o leitor."			2
"Atualiza o número de atrasos do leitor."		2	3
Commit	2	3	4

Figura 4.13 - Exemplo de uma TDBDA mista.

- Além dos valores V e F, as entradas de condição admitem o símbolo '-', significando que, para a regra em questão, a condição poderá ser V ou F. As figuras 4.14 e 4.15 ilustram uma TDBDA que utiliza o símbolo '-' e a mesma tabela sem este símbolo, respectivamente.

On DevolveLivroIME(VCodLivro, VNumDoc)			
$DtDevoluçãoLivroIME(VcodLivro, VNumDoc) = < Sysdate$	V	F	F
$NumAtrasosIME(VnumDoc)$	-	< 3	≥ 3
"Registra a Devolução do livro."	1	1	1
"Suspende o leitor."			2
"Atualiza o número de atrasos do leitor."		2	3
Commit	2	3	4

Figura 4.14 - Exemplo de uma TDBDA que utiliza o símbolo '-'.

<i>On DevolveLivroME(VCodLivro, VNumDoc)</i>				
<i>DtDevoluçãoLivroME(VcodLivro, VNumDoc) =< Sysdate</i>	V	V	F	F
<i>NumAtrasosME(VNumDoc) < 3</i>	V	F	V	F
"Registra a Devolução do livro."	1	1	1	1
"Suspende o leitor."				2
"Atualiza o número de atrasos do leitor."			2	3
<i>Commit</i>	2	2	3	4

Figura 4.15 - TDBDA da figura 4.14 sem o símbolo '!'.¹

- Uma **TDBDA** é **ambígua** quando apresenta ao menos duas regras situação-ação ambíguas, isto é, duas ou mais regras cuja avaliação de suas respectivas condições as tornam selecionáveis. As regras intencionais assinaladas na figura 4.16 são ambíguas e, consequentemente, a TDBDA também o é.

<i>On Reserva(VcodLivro, VNumDoc)</i>			
<i>EstáDisponível(VCodLivro)</i>	V	F	F
"Verifica se o livro VcodLivro está com o leitor VNumDoc"	-	-	F
"Reserva do livro"			1
<i>Msg('A reserva não pode ser feita.')</i>	1	1	
<i>Commit</i>			2
<i>Rollback</i>	2	2	

Figura 4.16 - TDBDA da figura 4.1 modificada, com duas regras ambíguas.

4.1.4 Completude de uma TDBDA

Uma das vantagens de utilizarmos as TDBDA's é a facilidade de verificarmos, sistematicamente, se todas as combinações possíveis das entradas de condições são cobertas para um determinado problema [Hum73] [Nag90].

Se todas estas combinações estão presentes, dizemos que a tabela em questão é completa. Mais formalmente, uma **TDBDA** é **completa** se toda valoração das entradas de condição contempla uma situação prevista nesta tabela.

A TDBDA da figura 4.15 é um exemplo de tabela completa.

4.1.5 Desmembramento e Aglutinação de TDBDA's

Nesta seção, apresentaremos um método de divisão e junção de TDBDA's, que chamaremos de desmembramento e aglutinação, respectivamente. Tal método permite o particionamento de um problema complexo em subproblemas tão independentes dos outros quanto possível. Esses

subproblemas são, então, implementados como TDBDA's que interagem através dos eventos externos, comentados no capítulo 2. Posteriormente, podemos juntar, ou melhor, aglutinar tais tabelas formando a TDBDA correspondente ao problema exposto inicialmente.

A princípio, podemos observar que este método facilita o desenvolvimento "top-down" das regras intencionais. Onde começamos com uma descrição do problema e refinamos tal descrição em níveis cada vez menores, até chegarmos a subproblemas ou módulos bem definidos, simples e, principalmente, administráveis.

Em consequência disso, as TDBDA's produzidas também apresentarão uma importante característica que facilita seu teste e manutenção: a modularidade.

Vejamos o desmembramento de uma TDBDA.

4.1.5.1 Desmembramento de uma TDBDA

Consideremos a TDBDA da figura 4.17.

<i>On Empréstimo(VcodLivro, VnumDoc)</i>			
<i>ExemplarStatus(VcodLivro)</i>	= 'C'	<> 'C'	<> 'C'
<i>NumExemplaresC(VcodLivro) = NumEmprestados(VcodLivro)</i>	-	V	F
<i>Fire VerificaReservas(VcodLivro, VNumDoc)</i>			1
<i>msg('Livro só para consulta')</i>	1		
<i>msg('Todos os exemplares deste livro estão emprestados')</i>		1	
<i>Rollback</i>	2	2	

Figura 4.17 - TDBDA que trata do empréstimo de livros a leitores do IME.

De acordo com o domínio de aplicação descrito no capítulo 5, ao qual está associada a TDBDA deste exemplo, percebemos a existência de *dois assuntos*; o primeiro cujo objetivo é verificar se o livro em questão pode ou não ser emprestado e o segundo, se há exemplares do mesmo livro para serem emprestados. Além disso, notamos que se um determinado livro não pode ser emprestado, então não há a necessidade de verificarmos se existem exemplares a serem emprestados na biblioteca.

Logo, poderíamos **desmembrar** a TDBDA deste exemplo em duas. A primeira contendo a linha de condição cuja descrição é "*ExemplarStatus(VcodLivro)*"; nesta, se o livro só pode ser consultado abortamos a transação, caso contrário, continuamos o processamento com a chamada à segunda tabela resultante do desmembramento, através do evento externo "*VerificaQtdLivros(VcodLivro, VNumDoc)*". Esta última será responsável pela verificação da quantidade de livros existentes para empréstimo (figura 4.18).

Figura 4.18 - TDBDA's resultantes do desmembramento da TDBDA da figura 4.17.

Cabe ressaltar, que o desmembramento de uma TDBDA não interfere na confluência das regras intencionais resultantes; pois a semântica de execução apresentada na seção 3.2.2 baseia-se no processamento concorrente destas regras desde que suas ações sejam consistentes. Garantindo, assim, a confluência.

Agora, vamos analisar o processo inverso, garantindo que o processo de desmembramento é correto e reversível.

4.1.5.2 Aglutinação de uma TDBDA

Tomemos as TDBDA's da figura 4.18 e vamos aglutiná-las. Mas antes, notemos que é a primeira TDBDA desta figura que dispara a execução da segunda TDBDA.

O processo de aglutinação de duas tabelas tem duas etapas:

- construímos uma terceira TDBDA cujo evento é o mesmo da TDBDA que dispara a execução da segunda tabela e cujas descrições de condição e ação são formadas pela união das descrições de condição e ação das TDBDA's a serem aglutinadas, excluindo o evento externo da segunda TDBDA.

On Empréstimo(VcodLivro, VnumDoc)			
ExemplarStatus(VcodLivro)			
NumExemplaresC(VcodLivro) = NumEmprestados(VcodLivro)			
Fire VerificaReservas(VcodLivro, VNumDoc)			
msg('Livro só para consulta')			
msg('Todos os exemplares deste livro estão emprestados')			
Rollback			

Figura 4.19 - TDBDA resultante da aplicação da primeira etapa do processo de aglutinação das TDBDA's da figura 4.18.

- combinamos as entradas de condições destas tabelas, a fim de obtermos a TDBDA final.

<i>On Empréstimo(VcodLivro, VNumDoc)</i>			
<i>ExemplarStatus(VcodLivro)</i>	= 'C'	<> 'C'	<> 'C'
<i>NumExemplaresC(VcodLivro) = NumEmprestados(VcodLivro)</i>	-	V	F
<i>Fire VerificaReservas(VcodLivro, VnumDoc)</i>			1
<i>msg('Livro só para consulta')</i>	1		
<i>msg('Todos os exemplares deste livro estão emprestados')</i>		1	
<i>Rollback</i>	2	2	

Figura 4.20 - TDBDA resultante da aplicação da segunda etapa do processo de aglutinação das TDBDA's da figura 4.18.

A fim de formamos a TDBDA da figura 4.20, repetimos todas as regras situação-ação da primeira TDBDA cujas entradas de ação não disparem a execução da segunda TDBDA e, atribuímos o valor '-' às entradas de condição correspondentes às descrições de condições da segunda TDBDA. Já as regras intencionais que disparam a execução da segunda TDBDA, combinamos suas entradas de condições com todas as entradas de condições da segunda TDBDA.

4.1.6 Tradução de TDBDA's para Gatilhos de Bancos de Dados

Até agora, discutimos as TDBDA's e seus principais benefícios. Entretanto, os computadores são incapazes de entender sua estrutura e, conseqüentemente, executar as regras intencionais por elas representadas. Assim, devemos construir processadores¹⁷ capazes de entender essas tabelas e traduzir suas condições e ações em alguma linguagem que possa ser entendida pelos computadores.

Nesta seção adaptaremos um dos métodos freqüentemente usado na tradução de tabelas de decisão¹⁸: o teste seqüencial.

Este método é o mais intuitivo e consiste, basicamente, em tomarmos o evento da TDBDA em questão, cada uma de suas regras e gerarmos o comando "CREATE OR REPLACE RULE".

¹⁷ Na literatura existe uma distinção entre os processadores que produzem código de baixo nível, processadores propriamente dito, e os que produzem código em uma linguagem de alto nível, chamados pré-processadores.

¹⁸ Para conhecer outros métodos, consulte o Apêndice B.

Exemplo:

Consideremos a TDBDA da figura 4.20. O código produzido pela aplicação deste método é o seguinte:

```
CREATE OR REPLACE RULE Emprestimo
WHEN Emprestimo(VCodLivro, VNumDoc)
IF ExemplarStatus(VCodLivro) = 'C' THEN
    Msg('Livro só para consulta. ');
    ROLLBACK;
ELSIF (ExemplarStatus(VCodLivro) <> 'C') AND
    (NumExemplares(VCodLivro)=NumEmprestados(VCodLivro)) THEN
    msg('Todos os exemplares deste livro estão emprestados. ');
    ROLLBACK;
ELSIF (ExemplarStatus(VCodLivro) <> 'C') AND
    (NOT(NumExemplares(VCodLivro)=NumEmprestados(VCodLivro))) THEN
    FIRE VerificaReservas(VCodLivro, VNumDoc);
END IF;
```

É interessante notarmos que o algoritmo de execução de regras intencionais confluentes pode ser diretamente aplicado às regras intencionais produzidas pela tradução das diversas TDBDA's.

4.2 Vantagens do Uso de TDBDA's sobre o Método Tradicional de Projeto de Regras Intencionais

O uso de TDBDA's no projeto de regras intencionais traz uma série de vantagens em relação aos métodos convencionais. São elas:

- ❑ nos métodos convencionais, freqüentemente, definimos qual conjunto de ações ocorrerão para um determinado evento se uma ou mais condições são satisfeitas. Já com as TDBDA's, listamos todas as possíveis condições e ações, associando conjuntos específicos de condições a ações, formando as regras situação-ação desejadas.

- ❑ todas as condições e ações associadas a um evento específico estão dispostas graficamente num mesmo local, facilitando o entendimento do inter-relacionamento entre estas e outras regras (essencial na verificação da terminação, por exemplo) e futuras manutenções;
- ❑ verificação automática das entradas de condição através da aglutinação e desmembramento da TDBDA em questão; impedindo, assim, combinações de entradas de condição incompletas e, até mesmo, impossíveis; e
- ❑ interface amigável.

ESTUDO DE CASO: MODELAGEM DA BIBLIOTECA DO IME/USP

5.1 Introdução

O propósito deste capítulo é apresentar a aplicação das TDBDA's a um exemplo do mundo real. Assim, escolhemos a Biblioteca “Carlos Benjamin de Lyra” do Instituto de Matemática e Estatística da USP como fonte para a obtenção de tal exemplo.

A seguir, descreveremos a modelagem do sistema para a biblioteca em questão, apresentando seus objetivos, suas principais funções e regras de negócio, bem como o diagrama de entidade relacionamento e o esquema de banco de dados associados. Em seguida, aplicaremos as TDBDA's às regras de negócio descritas e, finalmente, mostraremos um exemplo de desmembramento e aglutinação das TDBDA's resultantes deste estudo de caso.

5.2 Descrição Geral do Sistema

O fluxo de trabalho da Biblioteca do IME¹⁹ é composto por vários subsistemas, tais como análise para aquisição de novas obras, compras, catalogação e controle bibliotecário.

Em nosso estudo de caso, estaremos considerando apenas o subsistema referente ao controle bibliotecário. Tal subsistema permitirá que o(a) bibliotecário(a) tenha o completo controle do acervo e dos leitores da biblioteca, bem como dos empréstimos, consultas e reservas feitas pelos leitores (da Biblioteca do IME e das bibliotecas de outras unidades da USP) e pelo público em geral.

5.3 Principais Regras de Negócio

As regras de negócio, apresentadas a seguir, foram extraídas do Regulamento da Biblioteca do IME publicado no Diário Oficial de 14/12/1972 [DO72]. As mais relevantes ao nosso estudo de caso são:

¹⁹ Neste texto, nos referenciaremos à Biblioteca “Carlos Benjamin Lyra” do Instituto de Matemática e Estatística da USP como Biblioteca do IME.

- (R₁) Os periódicos, as obras de referência e as obras retiradas de circulação não estão sujeitas ao regime de empréstimo.
- (R₂) Caberá à Biblioteca colocar em regime especial de consulta e empréstimo as obras de leitura básica recomendadas em cursos, quando nesse sentido for solicitada por algum docente do IME.
- (R₃) A Biblioteca será franqueada ao público, para consultas, mediante identificação do consulente.
- (R₄) O livro retirado para consulta no recinto da Biblioteca deve ser devolvido ao bibliotecário no mesmo dia em que lhe for entregue.
- (R₅) O cartão de matrícula é pessoal e intransferível, sendo válido por um ano, para alunos e por dois anos, para docentes.
- (R₆) Para fins de empréstimo poderão ser feitas reservas de livros que não se encontrarem na Biblioteca.
- (R₇) Ao consulente não será permitida a reserva de obras que já se encontram em seu poder.
- (R₈) O prazo de empréstimo poderá ser renovado desde que a obra não tenha sido reservada por outro leitor.
- (R₉) Aos alunos de graduação, será facultada a retirada simultânea de até cinco livros pelo prazo de oito dias corridos.
- (R₁₀) Aos docentes e aos alunos de pós-graduação será facultada a retirada simultânea de até dez livros, pelo prazo de trinta dias corridos.
- (R₁₁) O aluno que não devolver à Biblioteca a obra retirada por empréstimo, no prazo estipulado, terá, em seu cartão individual, uma anotação “em atraso”, para cada dois dias de atraso.
- (R₁₂) Após três anotações “em atraso”, o aluno perderá, por sete dias, o direito ao empréstimo.
- (R₁₃) Os dias de férias escolares não serão contados para efeito de cumprimento da suspensão, mas serão computados para efeito de contagem dos dias de atraso na devolução.

Além destas regras de negócio estabelecidas no Regulamento da Biblioteca do IME, o subsistema de controle bibliotecário tem as seguintes:

(R₁₄) Os leitores das bibliotecas de outras unidades da USP poderão fazer reservas.

(R₁₅) Os leitores das bibliotecas de outras unidades da USP poderão retirar simultaneamente até dois livros pelo prazo de oito dias corridos.

5.4 O Diagrama de Entidade-Relacionamento da Biblioteca do IME

Na figura 5.1 apresentamos o Diagrama de Entidade-Relacionamento (E-R) que nos mostra, graficamente, a estrutura lógica geral do banco de dados do subsistema de controle bibliotecário da Biblioteca do IME.

5.5 O Esquema de Banco de Dados da Biblioteca do IME

A seguir apresentamos a lista das tabelas finais do banco de dados que armazena as informações do controle bibliotecário da Biblioteca do IME e suas respectivas colunas. Tais tabelas foram obtidas em dois passos; no primeiro, reduzimos o Diagrama E-R a tabelas e, em seguida, aplicamos o processo de normalização, ambos descritos em [KS93].

O esquema final do banco de dados é o seguinte:

- ❑ *LIVRO* (Código, Título, Editora, DataPublicação, LocalPublicação, NúmeroPáginas)
- ❑ *EXEMPLAR* (CódigoLivro, Número, DataChegada, Status)
- ❑ *AUTORLIVRO* (CódigoLivro, NomeAutor)
- ❑ *COMPRA* (CódigoLivro, NúmeroExemplar, Preço, Livraria)
- ❑ *DOAÇÃO* (CódigoLivro, Número, Doador)
- ❑ *PERIÓDICO* (Título, Volume, Número, Ano)
- ❑ *ASSINATURA* (Título, Editora, DataEntrega, Início, Fim)
- ❑ *PERMUTA* (Título, Instituição, Permutado, Frequência)
- ❑ *INSTITUIÇÃO* (Nome, Endereço)
- ❑ *EVENTO* (Nome, Local, Data)
- ❑ *BIBLIOTECA* (Código, Nome, Unidade, Endereço)
- ❑ *LEITORIME* (NúmeroDocumento, Nome, Endereço, TipoDocumento, Classe, DataInscrição, NúmeroAtrasos)
- ❑ *LEITOROUTRABIBLIOTECA* (NúmeroDocumento, NomeBiblioteca, NomeLeitor, TipoDocumento, NúmeroAtrasos)
- ❑ *LEITORPÚBLICO* (NúmeroDocumento, Nome, TipoDocumento)
- ❑ *EMPRÉSTIMOIME* (CódigoLivro, NúmeroExemplar, NúmeroDocumentoLeitor, DataEmpréstimo, DataDevolução, DataEntregou)

- ❑ *EMPRÉSTIMOOUTRASBIBL* (CódigoLivro, NúmeroExemplar, NúmeroDocumentoLeitor, CódigoBiblioteca, DataEmpréstimo, DataDevolução, DataEntregou)
- ❑ *CONSULTAPERIÓDICO* (Título, NúmeroDocumentoLeitor, Data, DataEntregou)
- ❑ *CONSULTALIVRO* (CódigoLivro, NúmeroExemplar, NúmeroDocumentoLeitor, Data, DataEntregou)
- ❑ *RESERVA* (CódigoLivro, NúmeroDocumentoLeitor, DataReserva)
- ❑ *SUSPENSÃO* (NúmeroDocumentoLeitor, DataSuspensão, NúmeroDias)
- ❑ *EDITORAASSINATURA* (Nome do Atributo, NomeEditora, Endereço)

5.6 Representação das Regras de Negócio Através de TDBDA's

A seguir, apresentaremos as TDBDA's que implementam as principais regras de negócio do nosso estudo de caso. As TDBDA's apresentadas aqui não estão completas, pois preferimos deixar os detalhes de implementação para o apêndice C; neste apêndice também mostramos o código de todas as funções mencionadas abaixo.

5.6.1. Regras de Negócio para a Consulta a Livros

<i>On ConsultaLivro(VcodLivro, VNumDoc)</i>		
<i>NumExemplaresTotal(VcodLivro) = NumEmprestados(VcodLivro) + NumLivrosConsultados(VcodLivro)</i>	<i>V</i>	<i>F</i>
<i>"Registra a consulta ao livro."</i>		<i>1</i>
<i>msg('Não há exemplares para consulta.')</i>	<i>1</i>	
<i>Commit</i>		<i>2</i>
<i>Rollback</i>	<i>2</i>	

Figura 5.2 - TDBDA com as regras de negócio referentes à consulta a livros.

As funções mencionadas na TDBDA acima realizam as seguintes tarefas:

- ❑ *NumExemplaresTotal (VCodLivro IN NUMBER)*: retorna a quantidade de exemplares disponíveis na biblioteca.
- ❑ *NumEmprestados (VCodLivro IN NUMBER)*: retorna a quantidade de exemplares do livro *VCodLivro* que estão emprestados.
- ❑ *NumLivrosConsultados (VCodLivro IN NUMBER)*: retorna a quantidade de exemplares do livro *VCodLivro* que estão sendo consultados.

Figura 5.2 - Diagrama E-R para Controle Bibliotecário da Biblioteca do IME

5.6.2. Regras de Negócio para a Consulta a Periódicos

<i>On ConsultaPeriódico(VTitulo, VNumDoc)</i>		
<i>NumPeriódicos(VTitulo) = NumPeriódicosCons(VTitulo)</i>	V	F
"Registra a consulta ao periódico"		1
<i>msg('Não há exemplares para consulta.')</i>	1	
<i>Commit</i>		2
<i>Rollback</i>	2	

Figura 5.3 - TDBDA com as regras de negócio referentes à consulta a periódicos.

As funções mencionadas na TDBDA acima realizam as seguintes tarefas:

- *NumPeriodicos (VTitulo IN VARCHAR)*: retorna a quantidade de periódicos disponíveis na biblioteca.
- *NumPeriodicosCons (VTitulo IN NUMBER)*: retorna a quantidade de periódicos, com o título *VTitulo*, que estão sendo consultados.

5.6.3. Regras de Negócio para a Reserva de Livros

<i>On Reserva(VCodLivro, VNumDoc)</i>			
<i>EstáDisponível(VcodLivro)</i>	V	F	F
"Verifica se o livro VcodLivro está com o leitor VnumDoc"	-	V	F
"Reserva do livro"			1
<i>msg('A reserva não pode ser feita, pois o livro está disponível.')</i>	1		
<i>msg('A reserva não pode ser feita, pois o leitor está com o livro.')</i>		1	
<i>Commit</i>			2
<i>Rollback</i>	2	2	

Figura 5.4 - TDBDA com as regras de negócio referentes à Reserva de Livros.

A função citada na TDBDA acima realiza a seguinte tarefa:

- *EstaDisponivel (VCodLivro IN NUMBER)*: verifica se o livro, cujo código é igual a *VCodLivro*, está disponível para empréstimo.

5.6.4. Regras de Negócio para a Devolução de Livros

Neste item iremos considerar somente a devolução de livros emprestados aos leitores da Biblioteca do IME, visto que as regras de negócio para as devoluções efetuadas pelos leitores de outras bibliotecas da USP são análogas.

<i>On DevolveLivroIME(VCodLivre, VNumDoc)</i>			
<i>DtDevoluçãoLivroIME(VCodLivre, VNumDoc) =< Sysdate</i>	<i>V</i>	<i>F</i>	<i>F</i>
<i>NumAtrasosIME(VnumDoc)</i>	<i>-</i>	<i>< 3</i>	<i>>= 3</i>
<i>"Registra a Devolução do livro."</i>	<i>1</i>	<i>1</i>	<i>1</i>
<i>"Suspende o leitor."</i>			<i>2</i>
<i>"Atualiza o número de atrasos do leitor."</i>			<i>3</i>
<i>Commit</i>	<i>2</i>	<i>2</i>	<i>4</i>

Figura 5.5 - TDBDA com as regras de negócio referentes à devolução de livros emprestados aos leitores da Biblioteca do IME.

As funções mencionadas na TDBDA acima realizam as seguintes tarefas:

- ❑ *DtDevolucaoLivroIME (VCodLivre IN NUMBER)*: retorna a data prevista para a devolução do livro *VCodLivre* que está com o leitor *VNumDoc*.
- ❑ *NumAtrasosIME (VCodLivre IN NUMBER, VNumDoc IN NUMBER)*: atualiza e retorna o número de anotações "em atrasos" para o leitor *VNumDoc*.

5.6.5. Regras de Negócio para a Devolução de Livros Consultados

As regras de negócio para a devolução dos livros consultados e dos periódicos são análogas. Portanto, iremos discutir, neste item, somente as referentes a livros.

<i>On DevolveLivroCon(VCodLivre, VNumDoc)</i>			
<i>Sysdate >= (SELECT Data</i> <i>FROM ConsultaLivre</i> <i>WHERE CodigoLivre = VCodLivre</i> <i>AND DataEntregou = NULL)</i>	<i>V</i>	<i>F</i>	<i>F</i>
<i>NumAtrasosGeral(VCodLivre, VnumDoc)</i>	<i>-</i>	<i>< 3</i>	<i>>= 3</i>
<i>"Registra a devolução do livro."</i>	<i>1</i>	<i>1</i>	<i>1</i>
<i>"Suspende o leitor."</i>			<i>2</i>
<i>"Atualiza o número de atrasos do leitor."</i>			<i>3</i>
<i>Commit</i>	<i>2</i>	<i>2</i>	<i>4</i>

Figura 5.6 - TDBDA com as regras de negócio referentes à devolução de livros consultados pelos leitores.

A descrição da função citada na TDBDA acima é a seguinte:

- *NumAtrasosGera*(*VCodLivre* IN *NUMBER*, *VNumDoc* IN *NUMBER*): atualiza e retorna o número de anotações “em atrasos” para o leitor *VNumDoc*.

5.6.6. Regras de Negócio para o Empréstimo de Livros

As regras de negócio que implementam o empréstimo de livros aos leitores do IME são similares às regras de negócio para o empréstimo de livros aos leitores de outras unidades da USP. Assim, decidimos apresentar apenas as regras referentes ao primeiro caso.

On <i>EmprestimoIME</i> (<i>VCodLivre</i> , <i>VNumDoc</i>)									
<i>ExemplarStatus</i> (<i>VCodLivre</i>)	= 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
<i>NumExemplaresC</i> (<i>VCodLivre</i>) = <i>NumEmprestados</i> (<i>VCodLivre</i>)	-	V	F	F	F	F	F	F	F
<i>HaReservas</i> (<i>VCodLivre</i>)	-	-	V	V	V	V	V	V	V
<i>EstaNaReserva</i> (<i>VCodLivre</i> , <i>VNumDoc</i>)	-	-	F	F	V	V	V	V	V
<i>Classificação</i> (<i>VNumDoc</i>)	-	-	-	-	= 'p'	= 'p'	= 'p'	= 'p'	= 'p'
<i>DtInscrição</i> (<i>VNumDoc</i>)	-	-	-	-	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365
<i>EstaSuspensa</i> (<i>VNumDoc</i>)	-	-	-	-	V	F	F	F	F
<i>NumLivrosLeitor</i> (<i>VNumDoc</i>)	-	-	-	-	-	>= 10	>= 10	< 10	< 10
"Calcula a data de devolução do livro."									1
"Registra o empréstimo"									2
<i>ApagaReserva</i> (<i>VCodLivre</i> , <i>VcodLeitor</i>)									4
<i>msg</i> ('Livro só para consulta')	1								
<i>msg</i> ('Todos os exemplares deste livro estão emprestados')		1							
<i>msg</i> ('Livro reservado.')			1						
<i>msg</i> ('O leitor precisa se recadastrar.')									
<i>msg</i> ('O leitor está suspenso.')				1					
<i>msg</i> ('O leitor já estourou sua quota de livros.')							1		
<i>Commit</i>									3
<i>Rollback</i>	2	2	2	2	2	2			

Figura 5.7 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME.

On EmprestimoIME(VCodLivro, VNumDoc)						
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F	F	F
HaReservas(VCodLivro)	V	V	V	V	V	V
EstaNaReserva(VCodLivro, VNumDoc)	V	V	V	V	V	V
Classificacao(VNumDoc)	= 'P'	= 'D'	= 'D'	= 'D'	= 'G'	= 'G'
DInscricao(VNumDoc)	< Sysdate - 365	>= Sysdate - 730	>= Sysdate - 730	< Sysdate - 730	>= Sysdate - 365	>= Sysdate - 365
EstaSuspensa(VNumDoc)	-	V	F	-	V	F
NumLivrosLeitor(VNumDoc)	-	-	-	-	-	>= 4
"Calcula da data de devolucao do livro."			1			
"Registra o emprestimo"			2			
ApagaReserva(VCodLivro, VCodLeitor)			4			
msg('Livro só para consulta')						
msg('Todos os exemplares deste livro estão emprestados')						
msg('Livro reservado.')						
msg('O leitor precisa se recadastrar.')	1			1		
msg('O leitor está suspenso.')		1			1	
msg('O leitor já estourou sua quota de livros.')						1
Commit			3			
Rollback	2	2		2	2	2

Figura 5.8 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IMI: (Continuação).

On Emprestimo/ME(VCodLivro, VNumDoc)									
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F	F	F	F	F	F
HaReservas(VCodLivro)	V	V	V	V	V	V	V	V	V
EstaNaReserva(VCodLivro, VNumDoc)	V	V	V	V	V	V	V	V	V
Classificacao(VNumDoc)	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'
DtInscricao(VNumDoc)	>= Sysdate - 365	< Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	< Sysdate - 365
EstaSuspensa(VNumDoc)	F	-	-	-	-	-	-	-	-
NumLivrosLeitor(VNumDoc)	< 4	-	-	-	-	-	-	-	-
"Calcula da data de devolução do livro."	1								1
"Registra o empréstimo"	2								2
ApagaReserva(VCodLivro, VCodLeitor)	4								
msg('Livro só para consulta')									
msg('Todos os exemplares deste livro estão emprestados')									
msg('Livro reservado.')									
msg('O leitor precisa se recadastrar.')									
msg('O leitor está suspenso.')									
msg('O leitor já estourou sua quota de livros.')									
Commit	3								
Rollback									

Figura 5.9 - TTBDA com as regras de negócio para o empréstimo de livros aos leitores do IMI: (Continuação).

On EmpréstimoME(VCodLivro, VNumDoc)					
ExemplarStatus(VCodLivro)		<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmpréstados(VCodLivro)		F	F	F	F
HáReservas(VCodLivro)		F	F	F	F
EstáNaReserva(VCodLivro, VNumDoc)		-	-	-	-
Classificação(VNumDoc)		= 'D'	= 'D'	= 'D'	= 'G'
DthInscrição(VNumDoc)		>= Sysdate - 730	>= Sysdate - 730	< Sysdate - 730	>= Sysdate - 365
EstáSuspenso(VNumDoc)		V	F	-	V
NumLivrosLeitor(VNumDoc)		-	-	-	-
"Calcula da data de devolução do livro."			1		
"Registra o empréstimo"			2		
ApagaReserva(VCodLivro, VCodLeitor)					
msg('Livro só para consulta')					
msg('Todos os exemplares deste livro estão emprestados')					
msg('Livro reservado.')					
msg('O leitor precisa se recadastrar.')				1	
msg('O leitor está suspenso.')		1			1
msg('O leitor já estourou sua quota de livros.')					
Commit			3		
Rollback		2		2	2

Figura 5.10 - TTBIDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Continuação).

On EmprestimoME(VCodLivro, VNumDoc)				
ExemplarStatus(VCodLivro)		<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)		F	F	F
HaReservas(VCodLivro)		F	F	F
EstaNaReserva(VCodLivro, VNumDoc)		-	-	-
Classificação(VNumDoc)		= 'G'	= 'G'	= 'G'
DTInscrição(VNumDoc)		>= Sysdate - 365	>= Sysdate - 365	< Sysdate - 365
EstaSuspensa(VNumDoc)		F	F	-
NumLivrosLeitor(VNumDoc)		>= 4	< 4	-
"Calcula a data de devolução do livro."			1	
"Registra o empréstimo"			2	
ApagaReserva(VCodLivro, VCodLeitor)				
msg('Livro só para consulta')				
msg('Todos os exemplares deste livro estão emprestados')				
msg('Livro reservado.')				
msg('O leitor precisa se recadastrar.')				1
msg('O leitor está suspenso.')				
msg('O leitor já estourou sua quota de livros.')		1		
Commit			3	
Rollback		2		2

Figura 5.11 - T1DBDA com as regras de negócio para o empréstimo de livros aos leitores do IME (Continuação).

As funções mencionadas na TDBDA acima realizam as seguintes tarefas:

- *ExemplarStatus (VCodLivre IN NUMBER)*: retorna "C" se o livro só possui exemplares para consulta e "E" se o livro pode ser emprestado.
- *NumExemplaresC (VCodLivre IN NUMBER)*: retorna a quantidade de exemplares em circulação.
- *HaReservas (VCodLivre IN NUMBER)*: verifica se há reservas para o livro cujo código é *VCodLivre*.
- *EstaNaReserva (VCodLivre IN NUMBER, VNumDoc IN NUMBER)*: verifica se o leitor *VNumDoc* é o próximo leitor na reserva do livro *VCodLivre*.
- *Classificacao(VNumDoc IN NUMBER)*: retorna a classe do leitor *VNumDoc*.
- *DtInscricao(VNumDoc IN NUMBER)*: retorna a data de inscrição do leitor *VNumDoc* na Biblioteca do IME.
- *EstaSuspendido(VNumDoc IN NUMBER)*: verifica se o leitor *VNumDoc* está suspenso.
- *NumLivrosLeitorIME(VNumDoc IN NUMBER)*: retorna o número de livros que estão emprestados com o leitor *VNumDoc*.

5.7. Um Exemplo de Desmembramento e Aglutinação de TDBDA's

Nesta seção, apresentaremos um exemplo de desmembramento e, em seguida, aglutinação da TDBDA referente ao empréstimo de livros aos leitores da Biblioteca do IME (figuras 5.7 à 5.11).

5.7.1. Desmembramento de uma TDBDA

Neste exemplo de desmembramento procuramos originar módulos, isto é, TDBDA's que tratam de um único assunto referente ao empréstimo de livros aos leitores da Biblioteca do IME.

As TDBDA's resultantes do desmembramento da TDBDA em questão são:

<i>On Empréstimo(VCodLivro, VNumDoc)</i>		
<i>ExemplarStatus(VCodLivro)</i>	<i>= 'C'</i>	<i><> 'C'</i>
<i>Fire VerificaQtdLivros(VCodLivro, VNumDoc)</i>		<i>1</i>
<i>msg('Livro só para consulta')</i>	<i>1</i>	
<i>Rollback</i>	<i>2</i>	

Figura 5.12 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IME.

<i>On VerificaQtdLivros(VCodLivro, VNumDoc)</i>		
<i>NumExemplaresC(VcodLivro) = NumEmprestados(VCodLivro)</i>	<i>V</i>	<i>F</i>
<i>Fire VerificaReservas(VcodLivro, VnumDoc)</i>		<i>1</i>
<i>msg('Todos os exemplares deste livro estão emprestados')</i>	<i>1</i>	
<i>Rollback</i>	<i>2</i>	

Figura 5.13 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IME

<i>On VerificaReservas(VCodLivro, VNumDoc)</i>			
<i>HáReservas(VcodLivro)</i>	<i>V</i>	<i>V</i>	<i>F</i>
<i>EstaNaReserva(VCodLivro, VNumDoc)</i>	<i>V</i>	<i>F</i>	<i>-</i>
<i>Fire VerificaInscrição(VCodLivro, VNumDoc)</i>	<i>1</i>		<i>1</i>
<i>msg('Livro reservado.')</i>		<i>1</i>	
<i>Rollback</i>		<i>2</i>	

Figura 5.14 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IME.

On VerificaInscrição(VCodLivro, VnumDoc)					
Classificação(VnumDoc)	= 'P'	= 'P'	= 'D'	= 'D'	= 'G'
DthInscrição(VnumDoc)	>= Sysdate - 365	< Sysdate - 365	>= Sysdate - 730	< Sysdate - 730	>= Sysdate - 365
Fire VerificaSuspensão(VCodLivro, VnumDoc)	1		1		1
msg('O leitor precisa se recadastrar.')		1		1	1
Rollback		2		2	2

Figura 5.15 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IMI.

On VerificaSuspensão(VCodLivro, VnumDoc)		
EstáSuspensão(VnumDoc)	V	F
Fire VerificaLivrosLeitor(VCodLivro, VnumDoc)		1
Msg('O leitor está suspenso.')	1	
Rollback	2	

Figura 5.16 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IMI.

On VerificaLivrosLeitor(VCodLivro, VnumDoc)				
Classificação(VnumDoc)	= 'P'	= 'P'	= 'G'	= 'G'
NumLivrosLeitor(VnumDoc)	>= 10	< 10	>= 4	< 4
"Calcula da data de devolução do livro."		1		1
"Registra o empréstimo"		2		2
msg('O leitor já estourou sua quota de livros.')	1		1	
Rollback	2		2	
Commit		3		3

Figura 5.17 - TDBDA resultante do desmembramento da TDBDA que implementa as regras de negócio do empréstimo de livros aos leitores da Biblioteca do IMI.

5.7.2. Aglutinação das TDBDA's

Neste exemplo, aplicamos a aglutinação nas TDBDA's apresentadas nas figuras 5.12 à 5.17:

On Empréstimo(VCodLivro, VNumDoc)				
ExemplarStatus(VCodLivro)	= 'C'	<> 'C'	<> 'C'	
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	-	V	F	F
Fire VerificaReservas(VCodLivro, VNumDoc)				1
msg('Livro só para consulta')	1			
msg('Todos os exemplares deste livro estão emprestados')		1		
Rollback	2	2	2	

Figura 5.18 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12 e 5.13.

On Empréstimo(VCodLivro, VNumDoc)				
ExemplarStatus(VCodLivro)	= 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	-	V	F	F
HáReservas(VCodLivro)	-	-	V	F
EstaNaReserva(VCodLivro, VNumDoc)	-	-	V	-
Fire VerificaInscrição(VCodLivro, VNumDoc)			1	1
msg('Livro só para consulta')	1			
msg('Todos os exemplares deste livro estão emprestados')		1		
msg('Livro reservado.')			1	
Rollback	2	2	2	

Figura 5.19 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13 e 5.14.

On Empréstimo(VCodLivro, VNumDoc)					
ExemplarStatus(VCodLivro)	= 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	-	V	F	F	F
HáReservas(VCodLivro)	-	-	V	V	V
EstaNaReserva(VCodLivro, VNumDoc)	-	-	F	V	V
Classificação(VNumDoc)	-	-	-	= 'p'	= 'p'
DthInscrição(VNumDoc)	-	-	-	>= Sysdate - 365	< Sysdate - 365
Fire VerificaSuspensão(VCodLivro, VNumDoc)				1	
msg('Livro só para consulta')	1				
msg('Todos os exemplares deste livro estão emprestados')		1			
msg('Livro reservado.')			1		
msg('O leitor precisa se recadastrar.')					1
Rollback	2	2	2		2

Figura 5.20 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15.

On Empréstimo(VCodLivro, VNumDoc)					
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F	F
HáReservas(VCodLivro)	V	V	V	V	F
EstaNaReserva(VCodLivro, VNumDoc)	V	V	V	V	-
Classificação(VNumDoc)	= 'p'	= 'p'	= 'g'	= 'g'	= 'p'
DthInscrição(VNumDoc)	>= Sysdate - 730	< Sysdate - 730	>= Sysdate - 365	< Sysdate - 365	>= Sysdate - 365
Fire VerificaSuspensão(VCodLivro, VNumDoc)	1		1		1
msg('Livro só para consulta')					
msg('Todos os exemplares deste livro estão emprestados')					
msg('Livro reservado.')					
msg('O leitor precisa se recadastrar.')		1		1	
Rollback		2		2	

Figura 5.21 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 (continuação).

On Empréstimo(VCodLivro, VNumDoc)						
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F	F	F
HáReservas(VCodLivro)	F	F	F	F	F	F
EstaNaReserva(VCodLivro, VNumDoc)	-	-	-	-	-	-
Classificação(VNumDoc)	= 'P'	= 'D'	= 'D'	= 'D'	= 'G'	= 'G'
DtInscrição(VNumDoc)	< Sysdate - 365	>= Sysdate - 730	< Sysdate - 730	>= Sysdate - 365	< Sysdate - 365	< Sysdate - 365
Fire VerificaSuspensão(VCodLivro, VNumDoc)		1		1		
msg('Livro só para consulta')						
msg('Todos os exemplares deste livro estão emprestados')						
msg('Livro reservado.')						
msg('O leitor precisa se recadastrar.')	1		1			1
Rollback	2		2			2

Figura 5.22 - TTDIDA resultante da aglutinação das TTDIDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 (continuação).

On Empréstimo(VCodLivro, VNumDoc)									
ExemplarStatus(VCodLivro)		= 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)		-	V	F	F	F	F	F	F
HáReservas(VCodLivro)		-	-	V	V	V	V	V	V
EstaNaReserva(VCodLivro, VNumDoc)		-	-	F	V	V	V	V	V
Classificação(VNumDoc)		-	-	-	= 'p'	= 'p'	= 'p'	= 'p'	= 'p'
Diinscrição(VNumDoc)		-	-	-	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	< Sysdate - 365	< Sysdate - 365
EstáSuspendido(VNumDoc)		-	-	-	V	F	F	-	-
Fire VerificaLivrosLeitor(VCodLivro, VNumDoc)									
Msg('Livro só para consulta')		1					1		
Msg('Todos os exemplares deste livro estão emprestados')			1						
Msg('Livro reservado.')				1					
Msg('O leitor precisa se recadastrar.')									1
Msg('O leitor está suspenso.')					1				
Rollback		2	2	2	2	2	2	2	2

Figura 5.23 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16.

On Empréstimo(VCodLivre, VNumDoc)									
ExemplarStatus(VCodLivre)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplares(VCodLivre) = NumEmprestados(VCodLivre)	F	F	F	F	F	F	F	F	F
HáReservas(VCodLivre)	V	V	V	V	V	V	V	V	V
EstaNaReserva(VCodLivre, VNumDoc)	V	V	V	V	V	V	V	V	V
Classificação(VNumDoc)	= 'D'	= 'D'	= 'D'	= 'D'	= 'D'	= 'D'	= 'D'	= 'D'	= 'D'
DitInscrição(VNumDoc)	>= Sysdate - 730	>= Sysdate - 730	< Sysdate - 730	< Sysdate - 730	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	< Sysdate - 365	< Sysdate - 365
EstáSuspendido(VNumDoc)	V	F	-	-	V	V	F	-	-
Fire VerificaLivrosLeitor(VCodLivre, VNumDoc)		1					1		
msg('Livro só para consulta')									
msg('Todos os exemplares deste livro estão emprestados')									
msg('Livro reservado')									
msg('O leitor precisa se recadastrar')				1					1
msg('O leitor está suspenso')	1						1		
Rollback	2			2			2		2

Figura 5.24 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 (continuação).

On Empréstimo(VCodLivro, VNumDoc)									
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F	F	F	F	F	F
HáReservas(VCodLivro)	F	F	F	F	F	F	F	F	F
EstaNaReserva(VCodLivro, VNumDoc)	-	-	-	-	-	-	-	-	-
Classificação(VNumDoc)	= 'p'	= 'p'	= 'p'	= 'p'	= 'p'	= 'p'	= 'p'	= 'p'	= 'p'
DthInscrição(VNumDoc)	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	< Sysdate - 365	>= Sysdate - 730	>= Sysdate - 730	< Sysdate - 730	< Sysdate - 730
EstáSuspendido(VNumDoc)	V	V	F	F	-	V	F	-	-
Fire VerificaLivrosLeitor(VCodLivro, VNumDoc)			1	1			1		
msg('Livro só para consulta')									
msg('Todos os exemplares deste livro estão emprestados')									
msg('Livro reservado.')									
msg('O leitor precisa se recadastrar.')	1				1				1
msg('O leitor está suspenso.')	2				2				2
Rollback									

Figura 5.25 - ITDBDA resultante da aglutinação das ITDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 (continuação).

On Empréstimo(VCodLivro, VNumDoc)			
ExemplarStatus(VcodLivro)	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VcodLivro) = NumEmprestados(VCodLivro)	F	F	F
HáReservas(VCodLivro)	F	F	F
EstaNaReserva(VcodLivro, VNumDoc)	-	-	-
Classificação(VNumDoc)	= 'G'	= 'G'	= 'G'
Dthscricao(VNumDoc)	>= Sysdate - 365	>= Sysdate - 365	< Sysdate - 365
EstáSuspensa(VnumDoc)	V	F	-
Fire VerificaLivrosLeitor(VcodLivro, VNumDoc)		1	
msg('Livro só para consulta')			
msg('Todos os exemplares deste livro estão emprestados')			
msg('Livro reservado.')			
msg('O leitor precisa se recadastrar.')			1
msg('O leitor está suspenso.')	1		
Rollback	2		2

Figura 5.26 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 (continuação).

On EmprestimoME(VCodLivro, VNumDoc)		= 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
ExemplarStatus(VCodLivro)		-	V	F	F	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)		-	-	V	F	F
HaReservas(VCodLivro)		-	-	V	V	V
EstaNaReserva(VCodLivro, VNumDoc)		-	-	F	V	V
Classificação(VNumDoc)		-	-	-	= 'p'	= 'p'
DTinscrição(VNumDoc)		-	-	-	>= Sysdate - 365	>= Sysdate - 365
EstaSuspendido(VNumDoc)		-	-	-	F	F
NumLivrosLeitor(VNumDoc)		-	-	-	>= 10	< 10
"Calcula da data de devolução do livro."						1
"Registra o empréstimo"						2
Fire ApagaReserva(VCodLivro, VCodLeitor)						4
msg('Livro só para consulta')		1				
msg('Todos os exemplares deste livro estão emprestados')			1			
msg('Livro reservado.')				1		
msg('O leitor precisa se recadastrar.')						
msg('O leitor está suspenso.')					1	
msg('O leitor já estourou sua quota de livros.')						1
Commit		2	2	2	2	3
Rollback						

Figura 5.27 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 e 5.17.

On EmprestimoME(VCodLivro, VNumDoc)							
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplares(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F	F	F	F
HaReservas(VCodLivro)	V	V	V	V	V	V	V
EstaNaReserva(VCodLivro, VNumDoc)	V	V	V	V	V	V	V
Classificacao(VNumDoc)	= 'P'	= 'D'	= 'D'	= 'D'	= 'D'	= 'G'	= 'G'
DTInscricao(VNumDoc)	< Sysdate - 365	>= Sysdate - 730	>= Sysdate - 730	>= Sysdate - 730	< Sysdate - 730	>= Sysdate - 365	>= Sysdate - 365
EstaSuspensao(VNumDoc)	-	V	V	F	-	V	F
NumLivrosLeitor(VNumDoc)	-	-	-	-	-	-	>= 4
"Calcula da data de devolução do livro."				1			
"Registra o empréstimo"				2			
Fire ApagaReserva(VCodLivro, VCodLeitor)				4			
msg('Livro só para consulta')							
msg('Todos os exemplares deste livro estão emprestados')							
msg('Livro reservado.')							
msg('O leitor precisa se recadastrar.')	1				1		
msg('O leitor está suspenso.')		1				1	
msg('O leitor já estourou sua quota de livros.')							1
Commit				3			
Rollback	2	2			2	2	2

Figura 5.28 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 e 5.17 (Continuação).

On EmprestimoME(VCodLivre, VNumDoc)							
ExemplarStatus(VCodLivre)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplares(VCodLivre) = NumEmprestados(VCodLivre)	F	F	F	F	F	F	F
HaReservas(VCodLivre)	V	V	V	V	V	V	V
EstaNaReserva(VCodLivre, VNumDoc)	V	V	V	V	V	V	V
Classificação(VNumDoc)	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'
DthInscrição(VNumDoc)	>= Sysdate - 365	< Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	< Sysdate - 365
EstaSuspensão(VNumDoc)	F	-	-	V	F	F	-
NumLivrosLeitor(VNumDoc)	< 4	-	-	-	>= 10	< 10	-
"Calcula da data de devolução do livro."	1					1	
"Registra o empréstimo"	2					2	
Fire ApagaReserva(VCodLivre, VCodLeitor)	4						
msg('Livro só para consulta')							
msg('Todos os exemplares deste livro estão emprestados')							
msg('Livro reservado.')							
msg('O leitor precisa se recadastrar.')		1					1
msg('O leitor está suspenso.')			1				
msg('O leitor já estourou sua quota de livros.')				1			
Commit	3						
Rollback		2	2	2	2	3	2

Figura 5.29 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 e 5.17 (Continuação).

On EmprestimoIME(VCodLivro, VNumDoc)					
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F	F
HáReservas(VCodLivro)	F	F	F	F	F
EstaNaReserva(VCodLivro, VnumDoc)	-	-	-	-	-
Classificação(VNumDoc)	= 'D'	= 'D'	= 'D'	= 'D'	= 'G'
DtInscrição(VNumDoc)	>= Sysdate - 730	>= Sysdate - 730	< Sysdate - 730	< Sysdate - 730	>= Sysdate - 365
EstáSuspendo(VNumDoc)	V	F	-	-	V
NumLivrosLeitor(VNumDoc)	-	-	-	-	-
"Calcula da data de devolução do livro."		1			
"Registra o empréstimo"		2			
Fire ApagaReserva(VCodLivro, VcodLeitor)					
msg('Livro só para consulta')					
msg('Todos os exemplares deste livro estão emprestados')					
msg('Livro reservado.')					
msg('O leitor precisa se recadastrar.')				1	
msg('O leitor está suspenso.')	1				1
msg('O leitor já estourou sua quota de livros.')					
Commit		3			
Rollback	2			2	2

Figura 5.30 - TDBDA resultante da aglutinação das TDBDA's apresentadas nas figuras 5.12, 5.13, 5.14, 5.15 e 5.16 e 5.17 (Continuação).

Na seção 5.7.1, desmembramos a TDBDA representada nas figuras 5.7 à 5.11, originando as TDBDA's das figuras 5.12 à 5.17. Nesta seção, aglutinamos estas tabelas (figuras 5.18 à 5.30) e formamos a mesma TDBDA mostrada pelas figuras 5.7 à 5.11.

5.7.3. Considerações sobre Aglutinação e Desmembramento de TDBDA's: uma Experiência

A TDBDA considerada nos exemplos das duas seções anteriores foi a mais complexa para o desenvolvimento deste estudo de caso. Para que a mesma retratasse o empréstimo de livros aos leitores do IME, precisamos decompor tal problema em subproblemas que respondessem as seguintes questões:

- ☐ O livro pode ser emprestado?
- ☐ Há números de exemplares suficientes?
- ☐ Há reservas?
- ☐ Quem está na reserva?
- ☐ Qual o período de empréstimo para este leitor?
- ☐ O leitor não está suspenso e sua cota de livros ainda não estourou?

Desenvolvemos, então, uma TDBDA que respondesse cada uma delas e aglutinamo-as, derivando a TDBDA que implementa o empréstimo de livros aos leitores do IME.

Sem este método, o tratamento deste problema seria complexo e correríamos o risco de desenvolvermos uma TDBDA que não o retratasse, como aconteceu nos primórdios desta dissertação.

CONCLUSÃO

Os principais objetivos desta dissertação foram a apresentação do projeto de uma semântica confluyente para o processamento de regras intencionais e o desenvolvimento de um novo método de projeto destas regras utilizando as tabelas de decisão.

Para tanto, realizamos um amplo estudo sobre bancos de dados ativos e constatamos que estes são ideais para o desenvolvimento de determinados tipos de aplicação, como gerenciamento de restrições de integridade e sistemas especialistas. Entretanto, ainda haviam vários problemas em sua utilização, entre os quais a determinação do inter-relacionamento entre suas regras; que tratamos ao estendermos as tabelas de decisão a fim de que suportassem os eventos das regras intencionais. Além disso, propomos os métodos de desmembramento e aglutinação, que nos fornece uma excelente ferramenta para o desenvolvimento modular de TDBDA's. Tais métodos foram testados durante a concepção do estudo de caso apresentado no capítulo 5; onde nos deparamos com TDBDA's complexas, isto é, com um grande número de colunas de entradas de condição (como a TDBDA que retrata o empréstimo de livros aos leitores do IME e possui vinte e cinco colunas de entradas de condição), e cuja definição estaria sujeita a erros e/ou inconsistências se não utilizássemos a metodologia de desenvolvimento modular permitidas pelos métodos apresentados no capítulo 4.

Outro problema que verificamos no decorrer de nossos estudos foi o da confluência de regras intencionais, cuja origem é a escolha de uma regra intencional entre várias disparadas em um dado momento. Para resolvermos este problema, apresentamos uma semântica para o processamento de regras intencionais baseada na execução concorrente de múltiplas regras disparadas e no conceito de consistência de um conjunto de ações.

Cabe ressaltar que nosso estudo contemplou apenas duas das inúmeras questões relativas aos bancos de dados ativos, entre elas os problemas da terminação e do determinismo observável.

Em projetos futuros pretendemos expandir o tratamento de eventos de um dos Sistemas Gerenciadores de Banco de Dados Relacionais comerciais, como Oracle ou SQL Server que

trabalham apenas com os eventos "insert", "delete" e "update" em seus gatilhos ou "triggers"; implementar a semântica de execução de regras intencionais confluyente, apresentada no capítulo 3; desenvolver uma ferramenta CASE para TDBDA's que permita a tradução automática destas para gatilhos de bancos de dados; automatizar o desmembramento e aglutinação de TDBDA's; e, finalmente, projetar e implementar a paralelização das consultas pertencentes às TDBDA's desmembradas.

UMA LINGUAGEM PARA BANCO DE DADOS ATIVOS

Neste capítulo descreveremos as sintaxes da linguagem de consulta a bancos de dados SQL (“Structured Query Language”), da linguagem de programação PL/SQL (“Procedural Language/Structured Query Language”), ambas baseadas no SGBDR²⁰ Oracle 7.3, e da linguagem de regras intencionais desenvolvida exclusivamente para esta dissertação.

A.1 Linguagem de Consulta

A linguagem de consulta que apresentaremos aqui é inteiramente baseada na linguagem de consulta do SGDBR Oracle 7.3 e, conseqüentemente, 100% compatível com a SQL92, padrão definido pelos comitês ANSI/ISO.

A linguagem SQL tem diversas partes [KS93] [Orc96]:

- ❑ *DML* (“*Data Manipulation Language*” ou *Linguagem de Manipulação de Dados*): constituída da linguagem de consulta propriamente dita, baseada na álgebra relacional e no cálculo relacional de tupla, e de comandos para inserir, remover e modificar as tuplas de um banco de dados.
- ❑ *DDL* (“*Data Definition Language*” ou *Linguagem de Definição de Dados*): seus comandos permitem a definição, alteração e remoção dos objetos de um determinado esquema do banco de dados.
- ❑ “*Transaction Control*” (*Controle de Transações*): responsável pelo gerenciamento de mudanças feitas por um ou mais comandos DML.
- ❑ “*Session Control*” (*Controle de Sessões*): permite que as propriedades de uma determinada sessão do usuário sejam controladas.

²⁰ Sistema Gerenciador de Banco de Dados Relacional.

- ❑ “*Embedded SQL*” (*SQL Embutida*): segundo [KS93], a forma embutida da SQL é designada para dentro de linguagens de programação de uso geral como C, Fortran e Pascal.

A seguir, descreveremos as principais características dos comandos DML, de controle de transações e de uma linguagem de programação. Os leitores que desejarem se aprofundar nestes e nos demais tópicos devem consultar o “Oracle 7 Server Concepts Manual”²¹ e “Oracle 7 Server SQL Reference Manual”²² do SGBDR Oracle [Orc96].

A.1.1 Linguagem de Manipulação de Dados (DML)

Nesta seção discutiremos a sintaxe básica dos principais comandos DML da SQL:

- ❑ *Comando SELECT*: recupera informações do banco, ou melhor, das tabelas e visões de um banco de dados relacional. Sua sintaxe²³ básica é a seguinte:

```
SELECT [DISTINCT] {*, coluna [alias], ...}  
FROM tabela1 [, tabela2, ...]  
[WHERE condição (ões)]
```

onde:

- *SELECT*: é uma lista de, no máximo, uma coluna.
- *DISTINCT*: remove duplicatas.
- *coluna*: seleciona a coluna ou campo identificado.
- *alias*: atribui um cabeçalho diferente às colunas selecionadas.
- *FROM tabela1*: especifica a tabela que contém as colunas.
- *WHERE*: limita a consulta às linhas que satisfazem uma condição.
- *condição*: composta dos nomes de colunas, expressões, constantes e operadores de comparação.

²¹ Part Nº A32534-1.

²² Part Nº A32538-1

²³ Para obter maiores detalhes sobre expressões, constantes e operadores de comparações, consulte [Orc96].

Exemplo:

Consideremos a tabela EmprestimoIME do estudo de caso apresentado no capítulo 5. Vamos recuperar a data de devolução do livro cujo código é 001 e está emprestado ao leitor VNumDoc.

```
SELECT DataDevolucao
FROM EmprestimoIME
WHERE CodigoLivro = 001
AND NumeroDocumento = VNumDoc
AND DataEntregou = NULL;
```

- ❑ *Comando INSERT:* adiciona novas linhas a uma tabela. A sintaxe básica deste comando é:

```
INSERT INTO tabela [(coluna1 [, coluna2 ...])]
VALUES (valor1 [, valor2, ...])
```

onde:

- *tabela:* é o nome da tabela na qual os dados serão inseridos.
- *coluna:* é o nome da coluna da tabela a ser preenchida.
- *valor:* é o valor correspondente à coluna.

Exemplo:

Vamos inserir duas novas linhas à tabela EmprestimoIME.

```
INSERT INTO EmprestimoIME
VALUES (002, 01, 123456-78, '01/02/99', '15/02/99', NULL);

INSERT INTO EmprestimoIME (CodigoLivro, NumeroExemplar,
                           NumeroDocumentoLeitor, DataEmprestimo)
VALUES (003, 05, 123456-78, '01/02/99');
```

- ❑ *Comando UPDATE:* modifica as linhas existentes de uma determinada tabela. Sua sintaxe básica é:

```
UPDATE tabela
SET coluna1 = valor1 [, coluna2 = valor2 ...]
[WHERE condição];
```

onde:

- *tabela*: é o nome da tabela que terá seus dados alterados.
- *coluna*: é o nome da coluna que terá seu valor alterado.
- *valor*: é um valor ou sub-consulta correspondente à coluna mencionada acima.
- *condição*: identifica as linhas a serem atualizadas.

Exemplo:

Vamos registrar a data de devolução de um dos livros do exemplo anterior.

```
UPDATE EmprestimoIME
SET DataEntregou = sysdate 24
WHERE CodLivre = 002
AND NumeroDocumento = 123456-78
AND DataEntregou = NULL;
```

- ❑ **Comando DELETE:** remove uma ou mais linhas existentes em uma tabela. A sintaxe básica deste comando é:

```
DELETE [FROM] tabela
[WHERE condição];
```

onde:

- *tabela*: é o nome da tabela da qual será removida uma ou mais linhas.
- *condição*: identifica as linhas a serem removidas.

Exemplo:

Vamos remover todos os registros de empréstimos feitos pelo leitor cujo número do documento é 123456-78.

²⁴ *Sysdate* é uma função que retorna a data e hora do sistema.

DELETE EmprestimoIME

WHERE NumeroDocumento = 123456-78;

A.1.2 Controle de Transações

A unidade básica de processamento de banco de dados é a transação. E esta consiste de um ou mais comandos DML que alteram, de forma consistente, os dados deste banco de dados [KS93] [Orc96].

Há três comandos que controlam as transações, são eles:

- ❑ *Comando COMMIT*: termina a transação atual e transforma em permanente todos os dados pendentes. Sua sintaxe é:

COMMIT;

- ❑ *Comando SAVEPOINT*: marca um ponto dentro da transação atual, dividindo-a em seções menores. A sintaxe deste comando é:

SAVEPOINT nome;

onde:

- *nome*: identifica o “savepoint” a ser criado.

- ❑ *Comando ROLLBACK*: termina a transação atual e descarta todas as alterações de dados pendentes. Sua sintaxe é:

ROLLBACK [TO [SAVEPOINT] nome];

onde:

- *TO SAVEPOINT nome* ou *TO nome*: desfaz a transação corrente até o ponto especificado por nome.

A.2 Linguagem de Programação

Apesar de toda a funcionalidade e eficácia da SQL, seu objetivo é criar e manipular informações. No entanto, desenvolvedores de aplicação necessitam de mais; eles precisam ter a disposição estruturas de controle, variáveis, etc.

Assim, surgiu uma extensão da SQL, a PL/SQL, no SGBDR Oracle.

Nesta seção, abordaremos os seguintes tópicos: blocos de código básico, variáveis e constantes, tipos de dados, estruturas de controle e, finalmente, subprogramas.

A.2.1 Bloco PL/SQL

Um bloco PL/SQL é uma unidade composta de declarações e comandos, cujo principal objetivo é resolver um problema ou subproblema [Orc96].

Este bloco possui três seções, a saber:

- ❑ “*DECLARE*”: contém todas as variáveis, constantes, cursores e “exceptions” definidas pelo usuário, que serão referenciadas dentro da seção executável.
- ❑ “*BEGIN*”: contém comandos SQL para manipular dados no banco de dados e comandos PL/SQL para manipular dados no bloco.
- ❑ “*EXCEPTION*”: especifica as ações a serem realizadas quando surgem erros ou condições anormais dentro da seção executável.

Figura A.1 - Estrutura de um bloco PL/SQL. [Orc96].

Há vários tipos de blocos. Nesta dissertação abordaremos apenas os blocos simples e os subprogramas.

A.2.2 Variáveis e Constantes

Uma variável é uma área de memória a que se atribui um nome. Esta área nos permite armazenar e alterar valores dentro de um bloco.

Sua declaração é feita dentro da seção “*DECLARE*” do bloco PL/SQL e a sintaxe que nos permite fazê-la é:

identificador tipo [NOT NULL]
[:= | DEFAULT expr];

onde:

- *identificador*: é o nome do identificador
- *tipo*: tipo de dado escalar (*BINARY_INTEGER*, *NUMBER [(precisão, escala)]*, *CHAR [(tamanho_máximo)]*, *LONG*, *LONG RAW*, *VARCHAR2(tamanho_máximo)*, *DATE* e *BOOLEAN*) ou composto (tipos definidos pelo usuário).
- *NOT NULL*: restringe a variável de modo que contenha um valor.
- *expr*: qualquer expressão PL/SQL que possa seja um literal, outra variável ou uma expressão envolvendo operadores e funções.

Exemplo:

VGenero CHAR(01);
VCont BINARY_INTEGER := 0;

Cabe ressaltar que as variáveis são inicializadas utilizando o operador de atribuição *:=*.

Já uma constante é um tipo particular de variável cujo valor não pode ser alterado em tempo de execução. Para definirmos este tipo de variável, utilizamos a seguinte sintaxe:

identificador CONSTANT tipo := expr;

onde:

- *identificador, tipo e expr*: foram explicados anteriormente.
- *CONSTANT*: determina que a variável em questão é uma constante.

Exemplo:

CTaxaluros CONSTANT NUMBER(3,2) := 8.25

A.2.3 Tipos de Dados

Os tipos de dados determinam o formato de armazenamento, restrições e faixa de valores válidos de uma variável ou constante. A PL/SQL possui três categorias [Orc96]:

- ❑ *tipos de dados escalares*: aqueles que retêm um só valor e não tem componentes internos.

Os principais tipos de dados desta categoria são apresentados na figura A.2.

Tipo de Dados	Descrição
<i>BINARY_INTEGER</i>	<i>Números inteiros incluídos entre -2147483647 e +2147483647.</i>
<i>NUMBER[(precisão, escala)]</i>	<i>Números de ponto fixos e flutuantes.</i>
<i>CHAR[(tamanho_máximo)]</i>	<i>Cadeia de caracter de tamanho fixo de até 32767 bytes.</i>
<i>LONG</i>	<i>Cadeia de caracter de tamanho variável até 32760 bytes.</i>
<i>LONG RAW</i>	<i>Cadeia de dados binários de até 32760 bytes.</i>
<i>VARCHAR2(tamanho_máximo)</i>	<i>Cadeia de caracter de tamanho variável até 32767 bytes.</i>
<i>DATE</i>	<i>Datas e horas.</i>
<i>BOOLEAN</i>	<i>Armazena três possíveis valores: TRUE, FALSE E NULL.</i>

Figura A.2 - Principais tipos de dados escalares da PL/SQL.

- ❑ *tipos de dados compostos*: contém componentes internos e são reutilizáveis. A PL/SQL possui vários tipos compostos, como apresentado na figura A.3.

Tipo	Descrição	Sintaxe
<i>PL/SQL Table</i>	<p><i>Semelhante aos vetores das linguagens de 3ª Geração (3GL), devem conter dois componentes:</i></p> <ul style="list-style-type: none"> ❑ <i>uma chave primária do tipo BINARY_INTEGER, responsável pela indexação; e</i> ❑ <i>uma coluna do tipo escalar que armazena seus elementos.</i> 	<p><i>TYPE tipo IS TABLE OF tipo_escalar [NOT NULL] INDEX BY BINARY_INTEGER;</i></p> <p><i>Identificador tipo;</i></p>
	<p><i>Semelhante em estrutura a um registro de uma 3GL, isto é, formado por uma combinação de</i></p>	<p><i>TYPE tipo IS RECORD</i></p> <p><i>(campo1 tipo1 [NOT NULL {:= DEFAULT} expr,</i></p>

Tipo	Descrição	Sintaxe
<i>PL/SQL Record</i>	<i>outros tipos de dados.</i>	<i>[(campo2 tipo2 [NOT NULL {:= / DEFAULT} expr,...]); identificador tipo;</i>
<i>CURSOR</i>	<p><i>O SGBDR Oracle usa área de trabalho para o processamento de informações. A PL/SQL nos permite construir e manipular os dados contidos nestas áreas através dos cursores.</i></p> <p><i>Há dois tipos:</i></p> <ul style="list-style-type: none"> ☐ <i>implícito: declarados implicitamente pela PL/SQL para todos os comandos DML; e</i> ☐ <i>explícito: declarados pelo programador.</i> 	<pre> DECLARE ; CURSOR nome_cursor IS comando_select; </pre>

Figura A.3 - Tipos de dados compostos da PL/SQL.

- ☐ *tipos de dados definidos pelo usuário:* são tipos de dados personalizados, compostos por tipos de dados básicos e que mantêm as mesmas operações destes últimos. Na PL/SQL, a definição de um tipo definido pelo usuário é feita através da sintaxe a seguir:

SUBTYPE nome_subtype IS tipo_básico;

onde:

- *tipo_básico:* é um dos tipos analisados nos itens anteriores.

A.2.4 Estruturas de Controle

Na PL/SQL, e na maioria das linguagens de programação, o programa mais simples começa na instrução *BEGIN* do bloco de programa, executa cada instrução sequencialmente até chegar à instrução *END* [Obr92]. A figura A.4 ilustra esta estrutura de controle, cujo nome é **seqüência**.

Seqüência

Figura A.4 - Representação da estrutura de controle seqüencial [Orc96].

Entretanto, a maioria dos programas não podem ser expressos de maneira tão simples. Precisamos, portanto, de estruturas de controle que dêem aos programas a capacidade de agir de forma diferente em situações diferentes. Nesta seção avaliaremos dois outros tipos de estruturas de controle: condição e iteração (figura A.5).

Condição

Iteração

Figura A.5 - Representações das estruturas de controle condição e iteração, respectivamente [Orc96].

A.2.4.1 Condição

Condição ou desvio condicional é implementada pelo comando IF da PL/SQL. Tal comando permite que outras instruções sejam executadas seletivamente de acordo com o resultado da avaliação de uma condição.

A sintaxe deste comando é:

```
IF condição THEN
    Comandos;
[ELSIF condição THEN
    comandos;]
[ELSE
    comandos;]
END IF;
```

onde:

- *condição*: é uma variável ou expressão booleana.

Exemplo:

```
IF Vcontador > 0 THEN
    RETURN(TRUE);
ELSE
    RETURN(FALSE);
END IF;
```

A.2.4.2 Iteração

A iteração é a estrutura de controle que habilita os desenvolvedores a repetirem, em seus programas, uma sequência de comandos de acordo com a condição testada nesta estrutura.

Há três comandos que implementam esta estrutura de controle no PL/SQL, a saber:

- *LOOP*: neste comando, as instruções a serem repetidas são escritas entre as palavras chaves *LOOP* e *END LOOP*, como apresentado na sintaxe abaixo. Para controlarmos este comando, devemos ter uma condição de desvio que, no caso da PL/SQL, é a instrução *EXIT [WHEN condição]*.

```
LOOP
    comandos1;
    comandos2;
    :
EXIT [WHEN condição];
END LOOP;
```

Exemplo:

```
VSoma := 0;  
VMax := 10;  
VCont := 1;  
LOOP  
    VSoma := VSoma + VCont;  
    EXIT WHEN VCont > VMax;  
END LOOP;
```

- *FOR*: tem a mesma estrutura geral do comando *LOOP*, no entanto, o número de iterações é pré-definido. Sua sintaxe é a seguinte:

```
FOR indice IN [REVERSE] limite_inferior ... limite_superior LOOP  
    comandos1;  
    comandos2;  
    :  
END LOOP;
```

onde:

- *índice*: é um número inteiro declarado implicitamente cujo valor aumenta automaticamente ou diminui de um em um em cada iteração do loop até ser alcançado o limite superior.
- *REVERSE*: faz com que o índice diminua com cada iteração a partir do limite superior até o inferior.

Exemplo:

```
VSoma := 0;  
FOR VCont IN 1... 10 LOOP  
    VSoma := VSoma + VCont;  
END LOOP;
```

- *WHILE*: este comando repete uma sequência de instruções até a condição controladora não ser mais *TRUE*, observemos sua sintaxe:

```

WHILE condição LOOP
    comandos1;
    comandos2;
    :
END LOOP;

```

Exemplo:

```

VSoma := 0;
VCont := 1;
FOR VCont <= 10 LOOP
    VSoma := VSoma + VCont;
END LOOP;

```

A.2.5 Subprogramas

A divisão de um programa em pequenas partes, ou subprogramas, torna a sua criação, teste e manutenção mais simples; pois cada uma destas pequenas partes podem ser testadas independente do programa principal e, quando estiverem corretas, integradas ao mesmo.

O PL/SQL possui duas categorias de subprogramas, a saber:

- “*Procedure*”: é um subprograma que pode conter uma lista de parâmetros, mas não precisa retornar um valor. Sua sintaxe é:

```

PROCEDURE nome [(parâmetro, ...)]
IS
    bloco_PL/SQL;

```

onde:

- *nome*: é o nome da “procedure”.
- *parâmetro*: é a sintaxe de parâmetro mostrada no final deste item; e
- *bloco PL/SQL*: é o corpo da procedure que define a ação realizada pela procedure, e cuja sintaxe foi estudada no item 4.1.2.1.

Exemplo:

```
PROCEDURE EstaSuspenso(VNumDoc IN NUMBER, VContador OUT NUMBER)
/* Verifica se o leitor "VnumDoc" está suspenso. */
RETURN BOOLEAN
IS
    VOk BOOLEAN;
BEGIN
    SELECT COUNT(*)
    INTO VContador
    FROM Suspensao
    WHERE NumeroDocumento = VNumDoc
    AND SYSDATE BETWEEN DtSuspensao AND (DtSuspensao +
                                           NumDias);
END EstaSuspenso;
```

- “Function”: é um subprograma que deve retornar um valor, e é invocado como parte de uma expressão. A sintaxe desta categoria de subprogramas é a seguinte:

```
FUNCTION nome [(parâmetro, ...)]
RETURN tipo_de_dado
IS
    bloco_PL/SQL;
```

onde:

- *RETURN tipo_de_dado*: determina o tipo de dado a ser retornado pela função em questão; e
- *bloco PL/SQL*: dentro do bloco_PL/SQL, devemos incluir o comando *RETURN valor*, a fim da função retornar um valor do mesmo tipo do *tipo_de_dado* da declaração *RETURN*.

Exemplo:

```
FUNCTION DtInscricao(VNumDoc IN NUMBER)
/* Retorna a data de inscrição do leitor "VnumDoc" na Biblioteca do IME. */
RETURN DATE
IS
    VDtInscricao DATE;
BEGIN
    SELECT DataInscricao
    INTO VDtInscricao
    FROM LeitorIME
    WHERE NumeroDocumento = VNumDoc;

    RETURN VDtInscricao;
END DtInscricao;
```

Na sintaxe das duas categorias de subprogramas aqui apresentadas, notamos um novo elemento da linguagem PL/SQL: os parâmetros. A sintaxe deste novo elemento é:

nome_parâmetro [IN | OUT | IN OUT] tipo_de_dado
[{:= | DEFAULT} expr]

onde:

- *IN*: para um valor do ambiente de chamada ao subprograma;
- *OUT*: retorna um valor a partir da “procedure” para o ambiente de chamada; e
- *IN OUT*: passa um valor do ambiente de chamada a uma “procedure” e retorna um valor possivelmente diferente da “procedure” para o ambiente de chamada.

Exemplo:

```
VNumDoc IN NUMBER
VNumDoc IN NUMBER, VContador OUT NUMBER
```

A.2.6 Interagindo com o Oracle

A PL/SQL interage com o Oracle utilizando os mesmos comandos descritos nas seções A.1.1 e A.2.1, à exceção do comando “*SELECT*”.

Vejamos a modificação necessária no comando “*SELECT*”:


```

SELECT lista_select
INTO nome_da_variável/nome_do_registro
FROM tabela
[WHERE condição (ões)]

```

onde:

- *lista_select*: é uma lista de pelo menos uma coluna e pode incluir expressões SQL, funções ou funções de grupo;
- *nome_da_variável*: é a variável escalar para reter o valor recuperado;
- *nome_do_registro*: é um RECORD PL/SQL para reter valores recuperados;
- *tabela*: especifica o(s) nome(s) da(s) tabela(s) do banco de dados; e
- *condição*: é composto de nomes de colunas, expressões, restrições de integridade e operadores, incluindo variáveis e constantes da PL/SQL.

Exemplo:

```

SELECT DataInscricao
INTO VDtInscricao
FROM LeitorIME
WHERE NumeroDocumento = VNumDoc;

```

Note que a cláusula *INTO* contém as variáveis de saída para o recebimento dos dados e que o comando acima deve retornar exatamente uma linha, caso contrário ocorrerá um erro.

A.3 Linguagem de Regras Intencionais

A linguagem de regras intencionais, discutida nesta seção, é composta por três comandos que nos permite defini-las e manipula-las, a saber:

- ❑ *CREATE RULE*;
- ❑ *DROP RULE*, e
- ❑ *ALTER RULE*..

A.3.1 O Comando *CREATE RULE*

Este comando é usado para a criação de uma regra intencional. Sua sintaxe é a seguinte:

```
CREATE [OR REPLACE] RULE name  
WHEN evento  
[IF condição]  
THEN ação;
```

onde:

- *OR REPLACE*: recria a regra intencional se ela já existe;
- *nome*: é o nome da regra intencional a ser criada ou substituída;
- *evento*: especifica um ou mais eventos que dispararão a regra intencional. Nesta linguagem de regras, os eventos permitidos são “*insert on tabela*”, “*delete from tabela*”, “*update tabela*” e os eventos externos (abordados no final desta seção);
- *condição*: especifica os testes adicionais que serão feitos quando esta regra é disparada. Esta cláusula é idêntica à cláusula “*WHERE*” do comando “*SELECT*”; e
- *ação*: bloco PL/SQL contendo os comandos que serão executados quando a referida regra é disparada e sua condição avaliada como ‘V’.

Exemplo:

```
CREATE OR REPLACE RULE Emprestimo  
WHEN Emprestimo(VCodLivro, VNumDoc)  
IF ExemplarStatus(VCodLivro) = 'C' THEN  
    Msg('Livro só para consulta.');  
    ROLLBACK;  
ELSIF (ExemplarStatus(VCodLivro) <> 'C') AND  
    (NumExemplares(VCodLivro)=NumEmprestados(VCodLivro)) THEN  
    msg('Todos os exemplares deste livro estão emprestados.');  
    ROLLBACK;  
ELSIF (ExemplarStatus(VCodLivro) <> 'C') AND
```

```

        (NOT(NumExemplares(VCodLivre)=NumEmprestados(VCodLivre))) THEN
            FIRE VerificaReservas(VCodLivre, VNumDoc);
    END IF;

```

A.3.2 O Comando *DROP RULE*

Este comando remove uma regra intencional do banco de dados ativo. Sua sintaxe é a seguinte:

```
DROP RULE name;
```

Exemplo:

```
DROP RULE Emprestimo;
```

A.3.3 O Comando *ALTER RULE*

Nos permite habilitar ou desabilitar uma regra intencional. Para tanto, utilizamos a seguinte sintaxe:

```
ALTER RULE name {ENABLE | DISABLE};
```

onde:

- *ENABLE*: habilita uma regra intencional; e
- *DISABLE*: desabilita uma regra intencional.

Exemplo:

```
ALTER RULE Emprestimo DISABLE;
```

A.3.4 Outras Características

Além dos comandos apresentados nas seções anteriores, a sintaxe desta linguagem de regras intencionais possui algumas características que merecem destaque:

- ❑ eventos externos;
- ❑ valores de transição; e
- ❑ prioridades entre regras.

Os **eventos externos** são ideais para as regras intencionais que implementam sistemas pertencentes a um domínio de aplicação específico. Tais eventos possuem duas características diferenciais:

- sua parametrização; e
- serem disparados pela própria aplicação.

Sua sintaxe é:

FIRE nome_evento [(parâmetro, ...)] [: operação];

Exemplo:

Emprestimo (VCodLivre, VnumDoc);

Para dispara-los, emitimos o comando:

FIRE nome_evento [(parâmetro, ...)];

Exemplo:

FIRE Emprestimo (001, 123456-78);

Já os **valores de transição** são obtidos através dos parâmetros fornecidos pelos eventos externos.

Finalmente, não utilizamos **prioridades entre regras**, pois a semântica de execução de regras intencionais que apresentamos no capítulo 3 resolve o problema de termos mais de uma regra disparada em um dado momento.

Apêndice B

TABELAS DE DECISÃO

No final dos anos 60 surgiu a programação estruturada. Segundo V. Setzer [Set92], este paradigma está baseado em três conceitos: desenvolvimento “top-down” ou por refinamentos sucessivos, modularização e uso de três estruturas básicas – sequência de comandos (início ... fim), alternativas (se ... então ... senão e suas variações), e iterações (enquanto ... faça e variações).

Mas, apesar das inovações trazidas pela programação estruturada, continuamos a ter problemas no projeto, implementação e manutenção de programas. Isto se deve, principalmente, à distância entre a linguagem natural e as linguagens de programação. Gerald Weiberg [Wei71] expressa claramente esta dificuldade: “... quando conversamos com nossos computadores, infelizmente, na maioria das vezes, estamos falando línguas diferentes ...”.

Além disso, uma das estruturas da programação estruturada que, em alguns casos, tornava-se bastante complexa é a alternativa. Vejamos um simples exemplo extraído de [Obr92].

Exemplo:

Imagine que você está participando de um jogo de azar usando uma caixa cheia de bolas pretas e brancas. Um jogador pega duas bolas da caixa ao acaso e, dependendo da combinação de cores escolhidas, ele será pago na seguinte proporção:

<i>Primeira Bola</i>	<i>Segunda Bola</i>	<i>Rateio</i>
<i>Branca</i>	<i>Branca</i>	<i>0:1</i>
<i>Branca</i>	<i>Preta</i>	<i>2:3</i>
<i>Preta</i>	<i>Branca</i>	<i>1:1</i>
<i>Preta</i>	<i>Preta</i>	<i>2:1</i>

Para codificar este jogo usamos a seguinte estrutura:

```

Se PrimeiraBola = Branca então
  Se SegundaBola = Branca então
    Aposta ← 0;
  Senão
    Aposta ← Aposta * 2/3;
Senão
  Se SegundaBola = Branca então
    Aposta ← Aposta * 125;
  Senão
    Aposta ← Aposta * 2;

```

Apesar de simples, não é difícil imaginar quão complexo o exemplo acima poderia tornar-se, se acrescentarmos algumas bolas de cores diferentes.

As tabelas de decisão surgiram para ajudar o entendimento e, conseqüentemente, implementação e manutenção de estruturas alternativas ou regras de decisão complexas. Segundo [Nag90], seu principal objetivo era introduzir uma estruturação de algoritmos em nível mais alto do que o da programação estruturada.

O uso das tabelas de decisão iniciou-se em 1957 com o Projeto de Sistemas Integrados da “General Electric”. A partir deste projeto muitos trabalhos foram desenvolvidos, entre eles: O FORTAB, desenvolvido pela IBM e “Randy Corporation”; e o DETAB-X (“Decision Table, Experimental”). Em 1989, surgiu o “Logical Gem”, um pré-processador que traduz tabelas de decisão em linguagens como Pascal, C, FORTRAN e dBase.

É claro que esta ferramenta apresenta limitações; mas, neste trabalho, estaremos interessados apenas em suas características.

B.1 Regras de Decisão

As **regras de decisão** são sentenças da forma:

Se CONDIÇÃO então AÇÃO

onde *CONDIÇÃO* é uma expressão booleana e *AÇÃO* é o nome de uma ação ou de um conjunto delas.

²⁵ Só incluímos esta condição por questões didáticas.

Exemplo:

*Se SegundaBola = Branca então
Aposta $\leftarrow$ 0;*

Aqui a condição é SegundaBola = Branca e a ação, Aposta $\leftarrow$ 0.

Uma regra de decisão pode ser interpretada da seguinte forma:

"AÇÃO" será executada se "CONDIÇÃO" é verdadeira.

B.2 O Que é uma Tabela de Decisão?

Uma **tabela de decisão** é uma representação tabular das regras de decisão, isto é, uma forma tabular de apresentarmos as condições e as ações dos algoritmos.

Na literatura, existem várias formas para representarmos as tabelas de decisão, sendo que a mais utilizada é apresentada na figura B.1.

<i>PrimeiraBola = Branca</i>	<i>V</i>	<i>V</i>	<i>F</i>	<i>F</i>
<i>SegundaBola = Branca</i>	<i>V</i>	<i>F</i>	<i>V</i>	<i>F</i>
<i>Aposta $\leftarrow$ 0</i>	<i>1</i>			
<i>Aposta $\leftarrow$ Aposta * 2/3</i>		<i>1</i>		
<i>Aposta $\leftarrow$ Aposta * 1</i>			<i>1</i>	
<i>Aposta $\leftarrow$ Aposta * 2</i>				<i>1</i>
<i>Imprima("Você Perdeu!")</i>	<i>2</i>			

Figura B.1 - Uma das formas de representarmos as tabelas de decisão.

B.3 Estrutura e Interpretação das Tabelas de Decisão

Segundo Nagayama [Nag90], a tabela de decisão da figura B.1 é interpretada da seguinte forma:

- se *PrimeiraBola = Branca* e *SegundaBola = Branca*, então serão executados os comandos *Aposta $\leftarrow$ 0* e *Imprima("Você Perdeu!")*, respectivamente;
- se *PrimeiraBola = Branca* e *SegundaBola $\neq$ Branca*, então será executado o comando *Aposta $\leftarrow$ Aposta * 2/3*;

- se $PrimeiraBola \neq Branca$ e $SegundaBola = Branca$, então será executado o comando $Aposta \leftarrow Aposta * 1$;
- se $PrimeiraBola \neq Branca$ e $SegundaBola \neq Branca$, então será executado o comando $Aposta \leftarrow Aposta * 2$.

Observando a tabela de decisão da figura B.1 e sua interpretação, notamos que a diferença entre esta e uma tabela de decisão para banco de dados ativo (TDBDA) é a ausência da linha de evento.

Assim, as tabelas de decisão são divididas em duas partes: linhas de condição e de ação (figura B.2). Que por sua vez possuem duas seções cada: descrição e entradas de condições e descrição e entradas de ações (figura B.3).

$PrimeiraBola = Branca$	V	V	F	F
$SegundaBola = Branca$	V	F	V	F
$Aposta \leftarrow 0$	1			
$Aposta \leftarrow Aposta * 2/3$		1		
$Aposta \leftarrow Aposta * 1$			1	
$Aposta \leftarrow Aposta * 2$				1
$Imprima("Você Perdeu!")$	2			

Figura B.2 - Linhas de condição e ação de uma tabela de decisão.

Descrição das Condições	$PrimeiraBola = Branca$	V	V	F	F
	$SegundaBola = Branca$	V	F	V	F
	$Aposta \leftarrow 0$	1			
	$Aposta \leftarrow Aposta * 2/3$		1		
	$Aposta \leftarrow Aposta * 1$			1	
	$Aposta \leftarrow Aposta * 2$				1
Descrições de Ações	$Imprima("Você Perdeu!")$	2			

Figura B.3 – Descrições e entradas de condições e ações de uma tabela de decisão.

Cabe ressaltar que as entradas de ação também possuem números que indicam a ordem de sua execução na regra a qual faz parte.

B.4 Outras Características

Até agora, nos limitamos a apresentar as tabelas de decisão na sua representação mais simples. No entanto, estas tabelas detêm outras representações e características que as tornam mais poderosas, são elas:

- ❑ tabelas de decisão estendidas e mistas;
- ❑ condições dependentes e independentes;
- ❑ utilização dos símbolos 'S', '*' e '-';
- ❑ "else rule"; e
- ❑ regras e tabela de decisão ambíguas.

B.4.1 Tabelas de Decisão Estendidas e Mistas

A tabela de decisão que mostramos no início deste capítulo (figura B.1) é um bom exemplo da representação **limitada** destas tabelas. Nela, as descrições de todas as condições contêm uma expressão lógica com valor V ou F.

As descrições das condições de uma **tabela de decisão estendida** são expressões que devem ser complementadas com o conteúdo das respectivas entradas de condição para se tornar uma expressão lógica [Nag 90]. A figura B.4 é um exemplo deste tipo de tabela de decisão.

<i>PrimeiraBola</i>	<i>= Branca</i>	<i>= Branca</i>	<i>≠ Branca</i>	<i>≠ Branca</i>
<i>SegundaBola</i>	<i>= Branca</i>	<i>≠ Branca</i>	<i>= Branca</i>	<i>≠ Branca</i>
<i>Aposta ← 0</i>	<i>1</i>			
<i>Aposta ← Aposta * 2/3</i>		<i>1</i>		
<i>Aposta ← Aposta * 1</i>			<i>1</i>	
<i>Aposta ← Aposta * 2</i>				<i>1</i>
<i>Imprima("Você Perdeu!")</i>	<i>2</i>			

Figura B.4 - Exemplo de uma tabela de decisão estendida.

As **tabelas de decisão mistas** têm parte de suas descrições de condições com a das tabelas limitadas e parte como das tabelas estendidas (figura B.5).

<i>PrimeiraBola = Branca</i>	<i>V</i>	<i>V</i>	<i>F</i>	<i>F</i>
<i>SegundaBola</i>	<i>= Branca</i>	<i>≠ Branca</i>	<i>= Branca</i>	<i>≠ Branca</i>
<i>Aposta ← 0</i>	<i>1</i>			
<i>Aposta ← Aposta * 2/3</i>		<i>1</i>		
<i>Aposta ← Aposta * 1</i>			<i>1</i>	
<i>Aposta ← Aposta * 2</i>				<i>1</i>
<i>Imprima("Você Perdeu!")</i>	<i>2</i>			

Figura B.5 - Exemplos de tabela de decisão mista.

B.4.2 Condições Dependentes e Independentes

As condições de uma tabela de decisão podem ser **dependentes** ou **independentes**.

Segundo [Nag90], as condições dependentes são classificadas em dois tipos:

- *explicitamente dependentes*: se a descrição de ambas as condições operam sobre uma variável comum (figura B.6);

<i>X = 1</i>	<i>V</i>	<i>F</i>
<i>X = 2</i>	<i>F</i>	<i>V</i>
<i>A₁</i>	<i>1</i>	
<i>A₂</i>		<i>1</i>

Figura B.6 - Exemplo de condições explicitamente dependentes.

- *implicitamente dependentes*: se, embora, não operem sobre uma variável comum, a semântica do problema particular as torne inter-relacionadas (figura B.7).

<i>Sexo = Masculino</i>	<i>V</i>	<i>V</i>	<i>F</i>
<i>TemCertificadoReservista</i>	<i>V</i>	<i>F</i>	<i>F</i>
<i>A₁</i>	<i>1</i>		
<i>A₂</i>		<i>1</i>	
<i>A₂</i>			<i>1</i>

Figura B.7 - Exemplo de condições implicitamente dependentes.

Nesta figura, observamos que o fato de “*Sexo = Masculino*” ser falso implica que a pessoa não possui certificado de reservista; de acordo, é claro, com o domínio do problema em questão.

Caso as condições não sejam de nenhum destes tipos, dizemos que as mesmas são independentes.

B.4.3 Os Símbolos '\$', '*' e '-'

Além dos valores V e F, as entradas de condições admitem três outros símbolos ('\$ ', '*' e '-') que, na verdade, representam tais valores.

Os símbolos '\$' e '*', segundo [Nag90], determinam que a condição em questão não é importante para a regra de decisão que está sendo testada, mas se o teste for feito, tem o significado idêntico a V e F, respectivamente. Tais entradas são usadas para condições dependentes que, em consequência da avaliação de outras condições, são verdadeiras ou falsas.

Exemplo:

Consideremos a tabela de decisão da figura B.7. Podemos observar se "Sexo = Feminino", então a Segunda condição desta coluna de condições será falsa. Por outro lado, se TemCertificadoReservista é verdadeiro, então a primeira condição desta tabela será verdadeira. Logo, utilizando os símbolos '\$' e '' temos a seguinte tabela de decisão:*

<i>Sexo = Masculino</i>	\$	V	F
<i>TemCertificadoReservista</i>	V	F	*
<i>A₁</i>	1		
<i>A₂</i>		1	
<i>A₂</i>			1

Figura B.8 - Exemplo de utilização dos símbolos '\$' e '*'.

Já uma entrada de condição com o símbolo '-' significa que, para a regra em questão, a condição poderá ser V ou F.

Exemplo:

Vamos modificar o exemplo dado no início deste capítulo de tal forma que o jogador seria pago na proporção mostrada a seguir:

<i>Primeira Bola</i>	<i>Segunda Bola</i>	<i>Rateio</i>
<i>Branca</i>	<i>Branca</i>	<i>0:1</i>
<i>Branca</i>	<i>Preta</i>	<i>0:1</i>
<i>Preta</i>	<i>Branca</i>	<i>1:1</i>
<i>Preta</i>	<i>Preta</i>	<i>2:1</i>

Observamos que se a primeira bola for branca, o jogador perderá independente da cor da segunda bola. Assim, usando o símbolo '-' obtemos a seguinte tabela de decisão:

$PrimeiraBola = Branca$	V	F	F
$SegundaBola = Branca$	$-$	V	F
$Aposta \leftarrow 0$	1		
$Aposta \leftarrow Aposta * 2/3$			
$Aposta \leftarrow Aposta * 1$		1	
$Aposta \leftarrow Aposta * 2$			1
$Imprima("Você Perdeu!")$	2		

Figura B.9 - Exemplo de utilização do símbolo '!'.

B.4.4 “Else Rule”

Segundo [Nag90], a “else rule” é uma regra especificada nas tabelas de decisão que será selecionada sempre que nenhuma das regras $R_1, R_2, \dots, R_n$ é selecionada.

Há vários tipos de aplicações para este tipo de regra. No entanto, a mais intuitiva é a utilizarmos no tratamento de erros, conforme apresentado no exemplo abaixo.

				<i>Else</i>
C_1	V	V	F	
C_2	F	V	F	
A_1	1		1	
A_2		1	2	
<i>Erro</i>				1

Figura B.10 - Exemplo de "else rule".

B.4.5 Regras e Tabelas de Decisão Ambíguas

Dado um conjunto de regras de decisão, podemos analisá-los quanto sua ambigüidade ou não. Para tanto, basta verificarmos se existem duas ou mais regras cuja a avaliação de suas respectivas condições as tornam selecionáveis simultaneamente. Tais **regras** são ditas **ambíguas**.

As regras de decisão da figura B.11 são ambíguas.

C_1	V	$-$
C_2	$-$	V
A_1	1	
A_2		1

Figura B.11 - Exemplo de Tabela de Decisão Ambígua.

Segundo [Nag90], uma **tabela de decisão é ambígua** se possuir regras ambíguas. Portanto, a tabela de decisão do exemplo acima é ambígua.

B.5 Completude de uma Tabela de Decisão

Dizemos que uma **tabela de decisão** é **completa** se todas as combinações possíveis das entradas de condições são cobertas por esta tabela.

A tabela de decisão da figura B.12 é um exemplo de tabela completa.

<i>PrimeiraBola = Branca</i>	<i>V</i>	<i>V</i>	<i>F</i>	<i>F</i>
<i>SegundaBola = Branca</i>	<i>V</i>	<i>F</i>	<i>V</i>	<i>F</i>
<i>Aposta ← 0</i>	<i>1</i>			
<i>Aposta ← Aposta * 2/3</i>		<i>1</i>		
<i>Aposta ← Aposta * 1</i>			<i>1</i>	
<i>Aposta ← Aposta * 2</i>				<i>1</i>
<i>Imprima('Você Perdeu!')</i>	<i>2</i>			

Figura B.12 - Exemplo de Tabela de Decisão Completa.

Nem sempre é interessante termos tabelas de decisões completas, pois as mesmas podem torna-se onerosas ou não condizerem com os respectivos domínios de aplicação.

Exemplo:

Imagine que muitas das regras de uma determinada tabela executem as mesmas ações e possuam as mesmas prioridades.

Vejamos outro exemplo.

Exemplo:

*Consideremos a tabela de decisão da figura B.7. É óbvio que as condições *Sexo = Masculino* e *TemCertificadoReservista nunca*²⁶ assumirão os valores *F* e *V*, respectivamente. Logo não devemos considerá-los nesta tabela.*

Segundo [Hum73], existem duas formas de resolvemos os problemas mencionados nos exemplos acima:

- usar o símbolo ‘-’, indicando que o estado da condição em questão é irrelevante. No primeiro exemplo, as entradas de condição de todas as regras intencionais que executarem as mesmas ações seriam substituídas pelo símbolo ‘-’.

Neste método, cada símbolo ‘-’ pode ser desdobrado em duas regras.

²⁶ Segundo a Constituição em vigor.

Exemplo:

C_1	V	V	F	F
C_2	-	F	-	V
C_3	F	V	-	-
C_4	-	F	V	F

2 regras
4 regras
1 regra
4 regras +

11 regras

Figura B.13 - Número de Regras de uma Tabela de Decisão que Utiliza o Símbolo '-'.

O total possível de regras é igual a 16 e o número de regras representadas, 11. Portanto, a tabela de decisão acima não é completa.

- usar a regra “else” ou “else rule”. Neste caso, a tabela de decisão da figura B.7 poderia ser rescrita, originando a tabela da figura B.14.

	Else		
<i>Sexo = Masculino</i>	$\$$	V	
<i>TemCertificadoReservista</i>	V	F	

Figura B.14 - Uso da Regra "Else".

Uma desvantagem deste método é a impossibilidade de garantirmos a completude da tabela de decisão que o utiliza, pois ou as regras estão implicitamente representadas ou são cobertas pela “else rule”. Com a prática, notaremos que uma regra relevante poderá ser esquecida facilmente durante o desenvolvimento da aplicação em questão.

B.6 Divisão e Fatoração de Tabelas de Decisão

Neste item discutiremos, essencialmente, a fatoração de tabelas de decisão. Tal assunto será utilizado no final deste capítulo, quando discutiremos alguns métodos para a tradução destas tabelas.

Mas antes, vejamos o processo de divisão de uma tabela de decisão.

B.6.1 Divisão de Tabelas de Decisão

Segundo Nagayama [Nag90], para dividirmos uma tabela de decisão T – com condições $C_1, C_2, \dots, C_m$; ações $A_1, A_2, \dots, A_l$ e regras $R_1, R_2, \dots, R_n$ – por uma condição C_p devemos tomar cada regra R_j e construir seus respectivos *conjuntos* $R_j^i = \{C_{1p}, C_{2p}, \dots, C_{ip}, C_{i+1p}, \dots, C_{mp}, A_{1p}, A_{2p}, \dots, A_{lp}\}$ com $1 \leq j \leq n$. Em seguida, formamos dois novos conjuntos com base em R_j^i , são eles $\mathcal{R}_V = \{R_j^i : C_{ij} = 'V', 'S' \text{ ou } 'L' \text{ em } T\}$ e $\mathcal{R}_F = \{R_j^i : C_{ij} = 'F', '*' \text{ ou } 'L' \text{ em } T\}$. A partir destes conjuntos, construímos duas tabelas:

- T_V^i , contendo as condições de $\mathcal{R}_V^i$ e suas respectivas ações e regras; e
- T_F^i formada analogamente.

Caso $\mathcal{R}_V^i \neq \emptyset$ e/ou $\mathcal{R}_F^i \neq \emptyset$, dizemos que essas duas tabelas são a divisão de T pela linha de condição C_p .

Exemplo:

Seja a seguinte tabela de decisão:

		R_1	R_2	R_3	R_4
C_1	<i>PrimeiraBola = Branca</i>	V	V	F	F
C_2	<i>SegundaBola = Branca</i>	V	F	V	F
	<i>Aposta $\leftarrow 0$</i>	1			
A_1	<i>Aposta $\leftarrow$ Aposta * 2/3</i>		1		
A_2	<i>Aposta $\leftarrow$ Aposta * 1</i>			1	
A_3	<i>Aposta $\leftarrow$ Aposta * 2</i>				1
A_4	<i>Imprima("Você Perdeu!")</i>	2			

Figura B.15 - Tabela de Decisão do Exemplo do Início do Capítulo.

Dividindo esta tabela pela condição C_2 (SegundaBola = Branca), obtemos os seguintes conjuntos:

- $R_1^2 = \{C_{11} = 'V'; A_1; A_2; A_3; A_4\}$
- $R_2^2 = \{C_{12} = 'V'; A_1; A_2; A_3; A_4\}$
- $R_3^2 = \{C_{13} = 'F'; A_1; A_2; A_3; A_4\}$
- $R_4^2 = \{C_{14} = 'F'; A_1; A_2; A_3; A_4\}$

Agora, podemos formar os conjuntos $\mathcal{R}_V$ e $\mathcal{R}_F$.

$$\square \mathcal{R}_{V'} = \{R_2, R_3\}$$

$$\square \mathcal{R}_F = \{R_2, R_3\}$$

E as respectivas tabelas de decisão:

<i>PrimeiraBola = Branca</i>	<i>V</i>	<i>F</i>
<i>Aposta ← 0</i>	<i>1</i>	
<i>Aposta ← Aposta * 2/3</i>		
<i>Aposta ← Aposta * 1</i>		<i>1</i>
<i>Aposta ← Aposta * 2</i>		
<i>Imprima("Você Perdeu!")</i>	<i>2</i>	

Figura B.16 - Tabela de Decisão T2V, resultante da divisão da tabela da figura B.15.

<i>PrimeiraBola = Branca</i>	<i>V</i>	<i>F</i>
<i>Aposta ← 0</i>		
<i>Aposta ← Aposta * 2/3</i>	<i>1</i>	
<i>Aposta ← Aposta * 1</i>		
<i>Aposta ← Aposta * 2</i>		<i>1</i>
<i>Imprima("Você Perdeu!")</i>		

Figura B.17 - Tabela de Decisão T2F, resultante da divisão da tabela da figura B.15.

Na prática, formamos a tabela $T_{V'}$ mantendo todas as regras de T com entradas de condição 'V', 'F' ou '-' na linha de condição C_i e retirando as regras com outras entradas de condição. Analogamente, podemos formar a tabela T_F , no entanto mantemos as entradas de condição 'F', '*' ou '-'.

B.6.2 Fatoração de Tabelas de Decisão por uma Linha de Condição

Seja T uma tabela de decisão limitada não trivial e $T_{V'}$ e T_F a divisão de T por C_p . Considere uma tabela de decisão T' com uma única condição e duas ações A_1 e A_2 , onde A_1 é uma chamada para a execução de $T_{V'}$ e A_2 , para T_F (figura B.18).

C_i	<i>V</i>	<i>F</i>
<i>Execute $T_{V'}$</i>	<i>1</i>	
<i>Execute T_F</i>		<i>1</i>

Figura B.18 - Tabela de Decisão T' .

Ao conjunto de T' , $T_{V'}$ e T_F , com execução iniciada em T' , denominamos fatoração de T pela linha de condição C_p .

A forma usual de representarmos esta fatoração é apresentada na figura B.19.

Figura B.19 - Forma usual de representarmos a Fatoração de Tabelas de Decisão.

Exemplo:

Considerando a tabela de decisão do exemplo anterior, teríamos:

B.7 Tradução de Tabelas de Decisão

Na literatura há vários métodos para a tradução de tabelas de decisão. No capítulo 4 adaptamos um deste métodos a fim de traduzirmos as TDBDA's em uma linguagem que pudesse ser entendida pelos atuais computadores. Nesta seção discutiremos outros três métodos aplicados às tabelas de decisão puras:

- "Switch Method";
- Método de Máscaras; e

- Método de Fatoração Sucessivas.

Vejamos cada um deles através de exemplos.

B.7.2 "Switch Method"

O "Switch Method" usa a técnica de linearização de uma matriz multidimensional para a seleção de regras [Nag90] [Hum73]. Este método foi descrito inicialmente por Veinott [Ve66a] e aperfeiçoado pelo mesmo autor em [Ve66b].

Neste método, consideremos uma tabela de decisão T com condições $C_1, C_2, \dots, C_m$ e entradas de condição $C_j, 1 \leq j \leq n$ e $1 \leq i \leq m$. Formamos, então, o conjunto $C_i = \{c_{i1}, c_{i2}, \dots, c_{in}\}$ de entradas distintas de condição da linha C_i cuja cardinalidade é dada por $n_i, 1 \leq i \leq m, n_i \leq n$. Agora, produzimos os seguintes comandos para cada condição C_i :

Se $C_i = c_{i1}$ então $l_i \leftarrow 0$;

Senão Se $C_i = c_{i2}$ Então $l_i \leftarrow 1$

;

Senão Se $C_i = c_{i, n_i}$ então $l_i \leftarrow n_i - 2$

Senão $l_i \leftarrow n_i - 1$

e geramos a variável I para a indexação da regra:

$$I = (n_2 \times \dots \times n_m) l_1 + (n_3 \times \dots \times n_m) l_2 + \dots + (n_m) l_{m-1} + l_m + 1$$

Isto é, produzimos um comando de desvio, onde o valor de I indica a I -ésima regra a ser selecionada.

Exemplo:

Seja a tabela de decisão da figura B.15, os conjuntos de entradas distintas para as condições C_1 e C_2 e suas respectivas cardinalidades são:

$$\square \quad C_1 = \{V', F'\} \text{ e } n_1 = 2; \text{ e}$$

$$\square \quad C_2 = \{V', F'\} \text{ e } n_2 = 2.$$

Produzimos, então, os comandos necessários a cada uma destas condições:

Se PrimeiraBola = Branca Então $I_1 \leftarrow 0$;

Senão $I_1 \leftarrow 1$;

Se SegundaBola = Branca Então $I_2 \leftarrow 0$

Senão $I_2 \leftarrow 1$;

O cálculo para indexação das regras é:

$$I = 2 * I_1 + I_2 + 1$$

E a escolha da regra é dada por:

Caso I

1: $Aposta \leftarrow 0$;

2: $Aposta \leftarrow * 2/3$;

3: $Aposta \leftarrow Aposta$;

4: $Aposta \leftarrow * 2$;

Assim se PrimeiraBola = Branca e SegundaBola = Preta, então as variáveis I_1 , I_2 e I terão os seguintes valores:

$$I_1 = 0, I_2 = 1 \text{ e } I = 2 * 0 + 1 + 1 = 2$$

E a ação $Aposta \leftarrow Aposta * 2/3$ será executada.

A princípio, este método funcionará para tabelas de decisão limitadas e completas; no entanto, podemos estendê-lo a fim de que o mesmo suporte tabelas estendidas e incompletas. Para maiores detalhes recomendamos [Nag90].

B.7.3 Método de Máscaras

Até agora nos limitamos às tabelas de decisões sem a regra “else” e cujas entradas de condição eram ‘V’ ou ‘F’. Mas, como procederemos se uma destas entradas for o símbolo ‘-’? Ou, se tivermos que tratar um regra “else”?

A resposta a estas questões veio em 1965 com o **método de máscaras** de regras para tabelas de decisão limitadas.

Segundo [Nag90], o método consiste na utilização de três estruturas básicas para cada tabela com m condições e n regras:

- o **vetor de estados** S com m elementos, um para cada condição:

$$S = (a_1, a_2, \dots, a_m), \text{ com } a_i = \begin{cases} 1, & \text{se } C_i = V \\ 0, & \text{se } C_i = F \end{cases}$$

- uma matriz de bits $M_{m \times n}$ denominada **matriz de máscaras**:

$$M_{ij} = \begin{cases} 1, & \text{se } C_{ij} = V \text{ ou } F \\ 0, & \text{caso contrário} \end{cases}$$

- outras matriz de bits $T_{m \times n}$ dita **matriz tabela**:

$$T_{ij} = \begin{cases} 1, & \text{se } C_{ij} = V \\ 0, & \text{caso contrário} \end{cases}$$

Tendo estas estruturas construídas, fazemos uma operação “and” elemento a elemento de S com a coluna j da matriz M , $1 \leq j \leq n$. Se o resultado desta operação é igual à j -ésima coluna de T , então a regra R_j correspondente é selecionada.

Exemplo:

Voltemos à tabela de decisão da figura B.1 e suponhamos que as condições PrimeiraBola = Branca e SegundaBola = Branca sejam F e V , respectivamente.

O vetor S será:

$$S = (0, 1)$$

Construindo as matrizes M e T teremos:

$$M = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

O resultado da operação “and” entre S e as colunas 1 e 2 da matriz M não são iguais às colunas 1 e 2 da matriz T , respectivamente. Mas, o produto booleano de S pela terceira coluna de M é igual à coluna 3 de T e, portanto, a regra R_3 será selecionada; como era esperado.

B.7.4 Método de Fatorações Sucessivas

Não é difícil perceber que os métodos apresentados anteriormente são custosos ou restritivos. Basta notarmos que o primeiro método restringe as tabelas de decisão a serem traduzidas; e no terceiro devemos construir uma ou mais matrizes e, posteriormente, percorrê-las.

O método de fatorações sucessivas, também conhecido como método por decomposição ou bifurcação, tem como produto de sua aplicação uma árvore de decisão. Basicamente, fazemos sucessivas fatorações na tabela original até que as tabelas finais sejam triviais. A seguir, aplicaremos este método à tabela de decisão da figura B.15.

Exemplo:

Fatorando a tabela da figura B.15 pela linha de condição C_1 , temos:

E agora, pelas subtabelas T'_V e T'_F :

No nosso exemplo, a árvore de decisão produzida é completa pois a tabela de decisão que a originou também o é. Caso esta tabela fosse incompleta e possuísse a “else rule”, existiriam nós sem filhos e quando buscássemos por filhos inexistentes acionaríamos a “else rule”.

TABELAS DE DECISÃO PARA BANCO DE DADOS ATIVOS DO ESTUDO DE CASO

C.1 Representação das Regras de Negócio Através de TDBDA's

A seguir, apresentaremos as TDBDA's que implementam as principais regras de negócio do nosso estudo de caso.

C.1.1 Regras de Negócio para a Consulta a Livros

<i>On ConsultaLivro(VCodLivro, VNumDoc)</i>		
<i>NumExemplaresTotal(VcodLivro) = NumEmprestados(VCodLivro) + NumLivrosConsultados(VcodLivro)</i>	<i>V</i>	<i>F</i>
<i>Insert Into ConsultaLivro Values (VCodLivro, VNumDoc, Sysdate, NULL);</i>		<i>1</i>
<i>msg('Não há exemplares para consulta.')</i>	<i>1</i>	
<i>Commit</i>		<i>2</i>
<i>Rollback</i>	<i>2</i>	

Figura C.1 - TDBDA com as regras de negócio referentes à consulta a livros.

Os códigos das funções que aparecem na figura C.1 são:

```

FUNCTION NumExemplaresTotal (VCodLivro IN NUMBER)
/* Retorna a quantidade de exemplares disponíveis na biblioteca. */
RETURN NUMBER
IS
    VNumExemplares NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO VNumExemplares
    FROM Exemplares
    WHERE CodigoLivro = VCodLivro;

    RETURN (VNumExemplares);
END NumExemplaresTotal;
    
```

```

FUNCTION NumEmprestados (VCodLivro IN NUMBER)
/* Retorna a quantidade de exemplares do livro "VCodLivro" que estão
   emprestados. */
RETURN NUMBER
IS
    VNumEmprestadosIME,
    VNumEmprestadosBibl NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO VNumEmprestadosIME
    FROM EmprestimoIME
    WHERE CodigoLivro = VCodLivro;

    SELECT COUNT(*)
    INTO VNumEmprestadosBibl
    FROM EmprestimoOutrasBibl
    WHERE CodigoLivro = VCodLivro;

    RETURN (VNumEmprestadosIME + VNumEmprestadosBibl);
END NumEmprestados;

FUNCTION NumLivrosConsultados (VCodLivro IN NUMBER)
/* Retorna a quantidade de exemplares do livro "VCodLivro" que estão
   sendo consultados. */
RETURN NUMBER
IS
    VNumLivrosConsultados NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO VNumLivrosConsultados
    FROM ConsultaLivro
    WHERE CodigoLivro = VCodLivro;

    RETURN (VNumLivrosConsultados);
END NumLivrosConsultados;

```

C.1.2 Regras de Negócio para a Consulta a Periódicos

On ConsultaPeriódico(VTítulo, VNumDoc)		
NumPeriódicos(VTítulo) = NumPeriódicosCons(VTítulo)	V	F
Insert Into ConsultaPeriódico Values (VTítulo, VNumDoc, Sysdate, NULL);		1
msg('Não há exemplares para consulta.')	1	
Commit		2
Rollback	2	

Figura C.2 - TDBDA com as regras de negócio referentes à consulta a periódicos.

Os códigos das funções que aparecem na figura C.2 são:

```
FUNCTION NumPeriodicos (VTitulo IN VARCHAR)
/* Retorna a quantidade de periódicos disponíveis na biblioteca. */
RETURN NUMBER
IS
    VNumPeriodicos NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO VNumPeriodicos
    FROM Periodico
    WHERE Titulo = VTitulo;

    RETURN (VNumPeriodicos);
END NumPeriodicos;

FUNCTION NumPeriodicosCons (VTitulo IN NUMBER)
/* Retorna a quantidade de periódicos, com o título VTitulo, que estão sendo
consultados. */
RETURN NUMBER
IS
    VNumPeriodicosCons NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO VNumPeriodicosCons
    FROM ConsultaPeriodico
    WHERE Titulo = VTitulo;

    RETURN (VNumPeriodicosCons);
END NumPeriodicosCons;
```

C.1.3 Regras de Negócio para a Reserva de Livros

On Reserva(VCodLivro, VNumDoc)			
EstáDisponível(VCodLivro)	V	F	F
VNumDoc in (SELECT NumeroDocumento FROM Empréstimo WHERE CodigoLivro = VCodLivro AND DataEntregou IS NULL) UNION (SELECT NumeroDocumento FROM EmprestimoOutrasBibl WHERE CodigoLivro = VCodLivro AND DataEntregou IS NULL)	-	V	F
Insert Into Reserva Values (VCodLivro, VnumDoc, Sysdate);			1
msg('A reserva não pode ser feita, pois o livro está disponível.')	1		
msg('A reserva não pode ser feita, pois o leitor está com o livro.')		1	
Commit			2
Rollback	2	2	

Figura C.3 - TDBDA com as regras de negócio referentes à Reserva de Livros.

O código da função que aparece na figura C.3 é:

```

FUNCTION EstaDisponivel (VCodLivro IN NUMBER)
/* Verifica se o livro, cujo código é igual a VCodLivro, está disponível para
empréstimo. */
RETURN BOOLEAN
IS
BEGIN
    IF NumExemplaresCirculacao > NumEmprestados THEN
        RETURN (TRUE);
    ELSE
        RETURN (FALSE);
    END IF;
END EstaDisponivel;

```

C.1.4 Regras de Negócio para a Devolução de Livros

Neste item iremos considerar somente a devolução de livros emprestados aos leitores da Biblioteca do IME, visto que as regras de negócio para as devoluções efetuadas pelos leitores de outras bibliotecas da USP são análogas.

<i>On DevolveLivroIME(VCodLivro, VNumDoc)</i>			
<i>DtDevolucaoLivroIME(VCodLivro, VNumDoc) =< Sysdate</i>	<i>V</i>	<i>F</i>	<i>F</i>
<i>NumAtrasosIME(VCodLivro, VNumDoc)</i>	<i>-</i>	<i>< 3</i>	<i>>= 3</i>
<i>UPDATE EmprestimoIME</i> <i>SET DataEntregou = Sysdate</i> <i>WHERE CodLivro = VCodLivro</i> <i>AND NumeroDocumento = VNumDoc</i> <i>AND DataEntregou = NULL</i>	<i>1</i>	<i>1</i>	<i>1</i>
<i>INSERT INTO Suspensao</i> <i>VALUES (VNumDoc, Sysdate, 7)</i>			<i>2</i>
<i>UPDATE LeitorIME</i> <i>SET NumAtrasos = 0</i> <i>WHERE NumeroDocumento = VNumDoc</i>			<i>3</i>
<i>Commit</i>	<i>2</i>	<i>2</i>	<i>4</i>

Figura C.4 - TDBDA com as regras de negócio referentes à devolução de livros emprestados aos leitores da Biblioteca do IME.

Os códigos das funções que aparecem na figura C.4 são:

```

FUNCTION DtDevolucaoLivroIME (VCodLivro IN NUMBER)
/* Retorna a data prevista para a devolução do livro "VCodLivro" que está com
o leitor "VNumDoc" */
RETURN DATE
IS
    VDtDevolucao DATE;
BEGIN
    SELECT DataDevolucao
    INTO VDtDevolucao
    FROM EmprestimoIME
    WHERE CodigoLivro = VCodLivro
        AND NumeroDocumento = VNumDoc
        AND DataEntregou = NULL;

    RETURN (VDtDevolucao);
END DtDevolucaoIME;

FUNCTION NumAtrasosIME (VCodLivro IN NUMBER,
VNumDoc IN NUMBER)
/* Atualiza e retorna o número de anotações "em atrasos" para o leitor
"VNumDoc" */
RETURN NUMBER
IS
    VNumAtrasos,
    VNumDiasAtraso NUMBER;
BEGIN
    VNumDiasAtrasos := Sysdate - DtDevolucaoIME(VCodLivro, VNumDoc);
    IF VNumDiasAtrasos > 0 THEN

```

```

UPDATE LeitorIME
SET NumAtrasos = NumAtrasos + VNumDiasAtrasos/2;
WHERE NumeroDocumento = VNumDoc;

END IF;

SELECT NumAtrasos
INTO VNumAtrasos
FROM LeitorIME
WHERE NumeroDocumento = VNumDoc;

RETURN (VNumAtrasos);
END NumAtrasosIME;

```

C.1.5 Regras de Negócio para a Devolução de Livros Consultados

As regras de negócio para a devolução dos livros consultados e dos periódicos são análogas. Portanto, iremos discutir, neste item, somente as referentes a livros.

On DevolveLivroCon(VCodLivro, VNumDoc)			
Sysdate >= (SELECT Data FROM ConsultaLivro WHERE CodigoLivro = VCodLivro AND DataEntregou = NULL)	V	F	F
NumAtrasosGeral(VCodLivro, VNumDoc)	-	< 3	>= 3
UPDATE ConsultaLivro SET DataEntregou = Sysdate WHERE CodLivro = VCodLivro AND NumeroDocumento = VNumDoc AND DataEntregou = NULL	1	1	1
INSERT INTO Suspensao VALUES (VNumDoc, Sysdate, 7)			2
IF EXIST (SELECT NumeroDocumento FROM LeitorIME WHERE NumeroDocumento = VNumDoc) THEN UPDATE LeitorIME SET NumAtrasos = 0 WHERE NumeroDocumento = VNumDoc ELSIF EXIST (SELECT NumeroDocumento FROM LeitorOutrasBibl WHERE NumeroDocumento = VNumDoc) THEN UPDATE LeitorOutrasBibl SET NumAtrasos = 0 WHERE NumeroDocumento = VNumDoc END IF			3
Commit	2	2	4

Figura C.5 - TDBDA com as regras de negócio referentes à devolução de livros consultados pelos leitores.

O código da função que aparece na tabela da figura C.5 é:

```
FUNCTION NumAtrasosGeral (VCodLivre IN NUMBER,  
    VNumDoc IN NUMBER)  
/* Atualiza e retorna o número de anotações "em atrasos" para o leitor  
   "VNumDoc" */  
    RETURN NUMBER  
IS  
    VNumAtrasos,  
    VNumDiasAtraso NUMBER;  
    VData Date;  
    VTipoLeitor CHAR := "G";  
BEGIN  
    SELECT Data  
    INTO VData  
    FROM ConsultaLivre  
    WHERE CodigoLivre = VCodLivre  
        AND DataEntregou IS NULL;  
  
    VNumDiasAtrasos := Sysdate - VData;  
  
    IF EXIST (SELECT NumeroDocumento  
             FROM LeitorIME  
             WHERE NumeroDocumento = VNumDoc) THEN  
        VTipoLeitor := "I";  
    ELSIF EXIST (SELECT NumeroDocumento  
                FROM LeitorOutrasBibl  
                WHERE NumeroDocumento = VNumDoc) THEN  
        VTipoLeitor := "B"  
    END IF;  
  
    IF VTipoLeitor = "I" THEN  
        IF VNumDiasAtrasos > 0 THEN  
            UPDATE LeitorIME  
            SET NumAtrasos = NumAtrasos + VNumDiasAtrasos/2;  
            WHERE NumeroDocumento = VNumDoc;  
        END IF;  
  
        SELECT NumAtrasos  
        INTO VNumAtrasos  
        FROM LeitorIME  
        WHERE NumeroDocumento = VNumDoc;  
    ELSIF VTipoLeitor = "B" THEN
```



```

IF VNumDiasAtrasos > 0 THEN
    UPDATE LeitorOutrasBibl
    SET NumAtrasos = NumAtrasos + VNumDiasAtrasos/2;
    WHERE NumeroDocumento = VNumDoc;
END IF;

SELECT NumAtrasos
INTO VNumAtrasos
FROM LeitorOutrasBibl
WHERE NumeroDocumento = VNumDoc;

END IF;

RETURN (VNumAtrasos);
END NumAtrasosGeral;

```

C.5.6. Regras de Negócio para o Empréstimo de Livros

As regras de negócio que implementam o empréstimo de livros aos leitores do IME são similares às regras de negócio para o empréstimo de livros aos leitores de outras unidades da USP. Assim, decidimos apresentar apenas as regras referentes ao primeiro caso.

On EmpréstimoIME(VCodLivro, VNumDoc)									
ExemplarStatus(VCodLivro)	= 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	-	V	F	F	F	F	F	F	F
HáReservas(VCodLivro)	-	-	V	V	V	V	V	V	V
EstaNaReserva(VCodLivro, VNumDoc)	-	-	F	F	V	V	V	V	V
Classificação(VNumDoc)	-	-	-	-	= 'P'	= 'P'	= 'P'	= 'P'	= 'P'
DiInscrição(VNumDoc)	-	-	-	-	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365
EstáSuspensão(VNumDoc)	-	-	-	-	V	F	F	F	F
NumLivrosLeitor(VNumDoc)	-	-	-	-	-	>= 10	>= 10	>= 10	< 10
SELECT DECODE(Classificação(VNumDoc) = 'G', Sysdate + 7, Sysdate + 30)									1
INTO VdtDevolução									
FROM Dual;									
INSERT INTO EmpréstimoIME									
VALUES (VCodLivro, VCodLeitor, Sysdate, DtDevolução, NULL);									2
ApagaReserva(VCodLivro, VCodLeitor)									4
msg('Livro só para consulta')	1								
msg('Todos os exemplares deste livro estão emprestados')		1							
msg('Livro reservado.')					1				
msg('O leitor precisa se recadastrar.')									
msg('O leitor está suspensão.')						1			
msg('O leitor já estourou sua quota de livros.')							1		
Commit									3
Rollback	2	2	2	2	2	2	2	2	

Figura C.6 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IME.

On EmprestimoME(VCodLivro, VNumDoc)									
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplares(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F	F	F	F	F	F
HáReservas(VCodLivro)	V	V	V	V	V	V	V	V	V
EstaNaReserva(VCodLivro, VNumDoc)	V	V	V	V	V	V	V	V	V
Classificação(VNumDoc)	= 'P'	= 'D'	= 'D'	= 'D'	= 'D'	= 'D'	= 'D'	= 'D'	= 'G'
DtInscrição(VNumDoc)	< Sysdate - 365	>= Sysdate - 730	>= Sysdate - 730	>= Sysdate - 730	< Sysdate - 730	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365
EstáSuspendido(VNumDoc)	-	V	F	F	-	V	V	V	F
NumLivrosLeitor(VNumDoc)	-	-	-	-	-	-	-	-	>= 4
SELECT DECODE(Classificação(VNumDoc) = 'G', Sysdate + 7, Sysdate + 30)			1						
INTO VDtDevolução									
FROM Dual;									
INSERT INTO EmprestimoME									
VALUES (VCodLivro, VCodLeitor, Sysdate, DtDevolução, NULL);			2						
ApagaReserva(VCodLivro, VCodLeitor)			4						
msg('Livro só para consulta')									
msg('Todos os exemplares deste livro estão emprestados')									
msg('Livro reservado.')									
msg('O leitor precisa se recadastrar.')	1				1				
msg('O leitor está suspenso.')		1					1		
msg('O leitor já estourou sua quota de livros.')									1
Commit			3						
Rollback	2	2			2		2	2	2

Figura C.7 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IMF: (Continuação).

On EmprestimoIME(VCodLivro, VNumDoc)						
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F	F	F
HaReservas(VCodLivro)	V	V	V	V	V	V
EstaNaReserva(VCodLivro, VNumDoc)	V	V	V	V	V	V
Classificacao(VNumDoc)	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'	= 'G'
DtInscricao(VNumDoc)	>= Sysdate - 365	< Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	>= Sysdate - 365	< Sysdate - 365
EstaSuspensa(VNumDoc)	F	V	F	F	F	-
NumLivrosLeitor(VNumDoc)	< 4	-	-	>= 10	< 10	-
SELECT DECODE(Classificacao(VNumDoc) = 'G', Sysdate + 7, Sysdate + 30) INTO VDtDevolucao FROM Dual;	1				1	
INSERT INTO EmprestimoIME VALUES (VCodLivro, VCodLeitor, Sysdate, DtDevolucao, NULL);	2				2	
ApagaReserva(VCodLivro, VCodLeitor)	4					
msg('Livro só para consulta')						
msg('Todos os exemplares deste livro estão emprestados')						
msg('Livro reservado.')						
msg('O leitor precisa se recadastrar.')						
msg('O leitor está suspenso.')						
msg('O leitor já estourou sua quota de livros.')						
Commit	3					
Rollback						

Figura C.8 - TDBDA com as regras de negócio para o empréstimo de livros aos leitores do IMI: (Continuação).

On EmprestimoIME(VCodLivro, VNumDoc)				
ExemplarStatus(VCodLivro)	<> 'C'	<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)	F	F	F	F
HaReservas(VCodLivro)	F	F	F	F
EstaNaReserva(VCodLivro, VNumDoc)	-	-	-	-
Classificação(VNumDoc)	= 'D'	= 'D'	= 'D'	= 'G'
DtInscrição(VNumDoc)	>= Sysdate - 730	>= Sysdate - 730	< Sysdate - 730	>= Sysdate - 365
EstaSuspensa(VNumDoc)	V	F	-	V
NumLivrosLeitor(VNumDoc)	-	-	-	-
SELECT DECODE(Classificação(VNumDoc) = 'G', Sysdate + 7, Sysdate + 30)		1		
INTO VDtDevolução				
FROM Dual;				
INSERT INTO EmprestimoIME				
VALUES (VCodLivro, VCodLeitor, Sysdate, DtDevolução, NULL);		2		
ApagaReserva(VCodLivro, VCodLeitor)				
msg('Livro só para consulta')				
msg('Todos os exemplares deste livro estão emprestados')				
msg('Livro reservado.')				
msg('O leitor precisa se recadastrar.')			1	
msg('O leitor está suspenso.')	1			1
msg('O leitor já estourou sua quota de livros.')				
Commit		3		
Rollback	2		2	2

Figura C.9 - TTBIDA com as regras de negócio para o empréstimo de livros aos leitores do IMI: (Continuação).

On EmprestimoME(VCodLivro, VNumDoc)				
ExemplarStatus(VCodLivro)		<> 'C'	<> 'C'	<> 'C'
NumExemplaresC(VCodLivro) = NumEmprestados(VCodLivro)		F	F	F
HaReservas(VCodLivro)		F	F	F
EstaNaReserva(VCodLivro, VNumDoc)		-	-	-
Classificacao(VNumDoc)		= 'G'	= 'G'	= 'G'
DtInscricao(VNumDoc)		>= Sysdate - 365	>= Sysdate - 365	< Sysdate - 365
EstaSuspensa(VNumDoc)		F	F	-
NumLivrosLeitor(VNumDoc)		>= 4	< 4	-
SELECT DECODE(Classificacao(VNumDoc) = 'G', Sysdate + 7, Sysdate + 30) INTO VDtDevolucao FROM Dual;			1	
INSERT INTO EmprestimoME VALUES (VCodLivro, VCodLeitor, Sysdate, DtDevolucao, NULL);			2	
ApagaReserva(VCodLivro, VCodLeitor)				
msg('Livro só para consulta')				
msg('Todos os exemplares deste livro estão emprestados')				
msg('Livro reservado.')				1
msg('O leitor precisa se recadastrar.')				
msg('O leitor está suspenso.')				
msg('O leitor já estourou sua quota de livros.')		1		
Commit			3	
Rollback		2		2

Figura C.10 - TIDBA com as regras de negócio para o empréstimo de livros aos leitores do IMI: (Continuação).

Os códigos das funções que aparecem na figura C.6 são:

```
FUNCTION ExemplarStatus (VCodLivre IN NUMBER)
/* Retorna "C" se o livro só possui exemplares para consulta e "E" se
o livro pode ser emprestado. */
RETURN CHAR
IS
VNumExemplares NUMBER;
BEGIN
SELECT COUNT(*)
INTO VNumExemplares
FROM Exemplar
WHERE CodigoLivre = VCodLivre
AND UPPER(Status) = "E";

IF VNumExemplares > 0 THEN
RETURN ("E");
ELSE
RETURN ("C");
END IF;
END ExemplarStatus;

FUNCTION NumExemplaresC (VCodLivre IN NUMBER)
/* Retorna a quantidade de exemplares em circulação. */
RETURN NUMBER
IS
VNumExemplaresC NUMBER;
BEGIN
SELECT COUNT(*)
INTO VNumExemplaresC
FROM Exemplar
WHERE CodigoLivre = VCodLivre
AND UPPER(Status) = "E";

RETURN (VNumExemplaresC);
END NumExemplaresC;

FUNCTION HaReservas (VCodLivre IN NUMBER)
/* Verifica se há reservas para o livro cujo código é "VCodLivre". */
RETURN BOOLEAN
IS
VNumReservas NUMBER;
BEGIN
SELECT COUNT(*)
INTO VNumReservas
FROM Reserva
WHERE CodigoLivre = VCodLivre;
```

```

        IF VNumReservas > 0 THEN
            RETURN (TRUE);
        ELSE
            RETURN (FALSE);
        END IF;
    END HaReservas;

    FUNCTION EstaNaReserva (VCodLivro IN NUMBER,
        VNumDoc IN NUMBER)
        /* Verifica se o leitor "VnumDoc" é o próximo leitor na reserva do livro
            "VCodLivro". */
        RETURN BOOLEAN
    IS
        VProximo NUMBER;
    BEGIN
        SELECT NumeroDocumento
        INTO VProximo
        FROM Reservas
        WHERE CodigoLivro = VCodLivro
            AND MIN(DtReserva);

        RETURN VProximo;

    END EstaNaReserva;

    FUNCTION Classificacao(VNumDoc IN NUMBER)
        /* Retorna a classe do leitor "VnumDoc". */
        RETURN CHAR
    IS
        VClasse CHAR;
    BEGIN
        SELECT Classe
        INTO VClasse
        FROM LeitorIME
        WHERE NumeroDocumento = VNumDoc;

        RETURN VClasse;

    END Classificacao;

```



```

FUNCTION DtInscricao(VNumDoc IN NUMBER)
/* Retorna a data de inscrição do leitor "VnumDoc" na Biblioteca do IME. */
RETURN DATE
IS
    VDtInscricao DATE;
BEGIN
    SELECT DataInscricao
    INTO VDtInscricao
    FROM LeitorIME
    WHERE NumeroDocumento = VNumDoc;

    RETURN VDtInscricao;

END DtInscricao;

FUNCTION EstaSuspendido(VNumDoc IN NUMBER)
/* Verifica se o leitor "VnumDoc" está suspenso. */
RETURN BOOLEAN
IS
    VContador NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO VContador
    FROM Suspendido
    WHERE NumeroDocumento = VNumDoc
    AND SYSDATE BETWEEN DtSuspendido AND (DtSuspendido +
    NumDias);

    IF VContador > 0 THEN
        RETURN(TRUE);
    ELSE
        RETURN(FALSE);
    END IF;

    RETURN VContador;

END EstaSuspendido;

```

```

FUNCTION NumLivrosLeitorIME(VNumDoc IN NUMBER)
/* Retorna o número de livros que estão emprestados com o leitor "VnumDoc". */
RETURN NUMBER
IS
    VNumLivrosLeitor NUMBER := 0;
BEGIN
    SELECT COUNT(*)
    INTO VNumLivrosLeitor
    FROM EmprestimoIME
    WHERE NumeroDocumento = VNumDoc
        AND DataEntregou IS NULL;

    RETURN VNumLivrosLeitor;

END NumLivrosLeitor;

```

Bibliografia

- [AWH92] A. Alken, J. Widow, and J. M. Hellerstern. *Behavior of Database Production Rules: Termination, Confluence, and Observable Determinism*. In Proceedings of The ACM SIGMOD International Conference on Management of Data, pages 59-68, San Diego, California, June 1992.
- [BNP91] P. McBrien, M. Niézette, D. Pantazis, A. H. Seltveit, U. Sundin, B. Theodoulidis, G. Tzallas, and R. Wohed. *A Rule Language to Capture and Model Business Policy Specifications*. In Proceedings of Third Nordic Conference on Advanced Information Systems Engineering, Springer - Verlag, May 1991.
- [BW95] E. Baralis, and J. Widow. *Better Static Rule Analysis for Active Database System*. Technical Report, December 1995.
- [DO72] *Diário Oficial da União*. Imprensa Nacional, 1972.
- [For84] R. Forsyth. *Expert System: Principles and Case Studies*. Chapman and Hall, London, 1984.
- [Hum73] E. Humby. *Program from Decision Tables*. Mcdonald, London, American Elsevier, New York, 1973.
- [KS93] H. F. Korth, and A. Silberschatz. *Sistema de Banco de Dados*. Makron Books, 2ª Edição, São Paulo, 1993.
- [Lon98] K. Lonen. *Oracle 8 DBA Handbook*. Osborne, McGraw-Hill, Berkeley, California, 1998.
- [Nag90] S. Nagayama. *Tabelas de Decisão e Implantação do Gerador IME*. São Paulo, Dezembro 1990.
- [Obr92] S. O'Brien. *Turbo Pascal 6 Completo e Total*. Makron Books, 1992.
- [Ocr96] *Oracle7 Documentation*. Oracle Corporation, 1996.
- [Pre92] S. R. Pressman. *Engenharia de Software*. Makron Books, 1992.
- [Set92] V. W. Setzer. *Programação Super-Estruturada e o Gerador de Programas I-M-E*. In Anais do VI Simpósio Brasileiro de Engenharia de Software, Gramado – RS, Novembro 1992.
- [Tan95] A.. K. Tanaka. *Banco de Dados Ativos*. In 10º Simpósio Brasileiro de Banco de Dados, Recife – PE, Outubro 1995.

[Ve66a] C. G. Veinott. *Programming Decision Tables in Fortran, Cobol or Algol*. In Comm. ACM, Vol 9., Num. 1, pages 31-35, January 1966.

[Ve66b] C. G. Veinott. *More on Programming Decision Tables*. In Comm. ACM, Letters to the Editor, Vol 9., Num. 7, page 485, July 1966.

[Wat86] D. A. Waterman. *A Guide to Expert Systems*. Addison-Wesley Publishing, 1986.

[WC96] J. Widow, and S. Ceri. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.