UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Luiz Carlos Camargo

RESTRIÇÕES DE INTEGRIDADE E REGRAS ATIVAS EM BANCOS DE DADOS DISTRIBUÍDOS

Dissertação submetida à Universidade Federal de Santa Catarina como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação

Murilo Silva de Camargo

Florianópolis, 10 / 2001

RESTRIÇÕES DE INTEGRIDADE E REGRAS ATIVAS EM BANCOS DE DADOS DISTRIBUÍDOS

Luiz Carlos Camargo

Esta Dissertação foi julgada adequada para a obtenção do título de Mestre em Ciência da Computação, Área de Concentração (Sistemas de Computação) e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação.

Prof. Dr. Fernando A. O. Gauthier (Coord. CPGCC)

Banca Examinadora

Prof. Dr. Murilo S. de Camargo (orientador)

Prof. Dr. Roberto Willrich

Prof. Dr. Rosvelter João Coelho da Costa

Dedico esta dissertação àqueles que de uma forma simples e honesta contribuíram para sua realização. Em especial à minha mãe que sempre me incentivou a buscar novos horizontes.

Agradecimentos

- Ao Profº Dr. Murilo S. de Camargo, meu orientador, que me acolheu e me conduziu ao longo dessa jornada.
- A minha família (esposa: Sidnéia, filhos: Lucas, Tiago e Jordana) que souberam compreender a minha ausência nos momentos em que me dedicava a esta tarefa.
- Ao programa de bolsa de capacitação da UNIPAR – Universidade Paranaense.
- A Deus e a Madre Maria Theodoro Voiron.

Sumário

LISTA DE FIGURAS x

LISTA DE TABELAS xii

LISTA DE ABREVIATURAS xiii

INTRODUÇÃO 01

CAPITULO 2 07

BANCOS DE DADOS DISTRIBUÍDOS 07

INTRODUÇÃO 07

2.1 APLICABILIDADE 08

2.2 DISTRIBUIÇÃO DE DADOS 09

2.3TRANSPARÊNCIA 13

2.4 PROBLEMAS DO CONTEXTO DISTRIBUÍDO 14

2.4.1 PROBLEMAS NO PROJETO DISTRIBUÍDO 15

2.4.2 GERENCIAMENTO DE TRANSAÇÕES DISTRIBUÍDAS 16

2.4.2.1 BLOQUEIO EM TRANSAÇÕES DISTRIBUÍDAS 16

2.4.2.2 CONTROLE DE CONCORRÊNCIA OTIMISTA EM TRANSAÇÕES DISTRIBUÍDA 17

2.4.2.3 *TIMESTEMPING* 18

2.4.3 EXECUÇÃO DE CONSULTAS DISTRIBUÍDAS 18

2.4.4 CONTROLE DE CONCORRÊNCIA 19

2.4.5 PROBLEMAS NO PROJETO DISTRIBUÍDO 20

2.4.5 PROBLEMAS NO PROJETO DISTRIBUÍDO 20

2.4.6 PROBLEMAS ADICIONAIS 20

2.4.7 RELAÇÃO ENTRE OS PRINCIPAIS PROBLEMAS 20

CONSIDERAÇÕES DO AMBIENTE DISTRIBUÍDO 21

CAPITULO 3 23

RESTRIÇÕES DE INTEGRIDADE 23

INTRODUÇÃO 23

3.1 DEFINIÇÕES DAS RESTRIÇÕES DE INTEGRIDADE PASSIVA 24

3.1.1 VIOLAÇÕES DA INTEGRIDADE PASSIVA 25

3.1.2 RESTRIÇÕES DE INTEGRIDADE ESTRUTURAL APLICADA AO MODELO RELACIONAL 27

3.2 CUMPRIMENTO DAS RESTRIÇÕES DE INTEGRIDADE SE MÂNTICAS..............29

3.2.1 RESTRIÇÕES PRÉ-OMPILADAS..............29

3.2.2 ASSERÇÕES..............30

3.2.3 COERÇÃO..............30

3.2.4 *TRIGGERS*..............32

CONSIDERAÇÕES DAS RESTRIÇÕES DE INTEGRIDADE..............33

CAPITULO 4..............34

RESTRIÇÕES DE INTEGRIDADE EM BANCOS DE DADOS DISTRIBUÍDOS..............34

INTRODUÇÃO..............34

4.1 ASSERÇÕES DISTRIBUÍDAS..............35

4.1.1 ASSERÇÕES INDIVIDUAIS..............36

4.1.2 ASSERÇÕES DIRECIONADAS..............37

4.1.3 ASSERÇÕES ENVOLVENDO GREGAÇÃO..............39

4.2 VERIFICAÇÃO GLOBAL DAS RESTRIÇÕES DE INTEGRIDADE DISTRIBUÍDAS..40

4.2.1 MÉTODO PARA VERIFICAÇÃO LOCAL DAS RESTRIÇÕES DE INTEGRIDADE..............40

4.2.2 GERENCIAMENTO DAS RESTRIÇÕES DE INTEGRIDADE EM PROJETOS DISTRIBUÍDOS DE BANCO DE DADOS..............42

CONCLUSÃO..............44

CAPITULO 5..............46

RESTRIÇÕES DE INTEGRIDADE DISTRIBUÍDAS NO INGRES..............46

INTRODUÇÃO..............46

5.1 REPLICAÇÃO DE DADOS NO INGRES..............46

5.1.1 POSSIBILIDADES DE REPLICAÇÃO NO INGRES..............49

5.1.2 TRATAMENTO DE COLISÃO NO INGRES..............51

5.2 DISTRIBUIÇÃO DE DADOS NO INGRES..............53

5.3 COMUNICAÇÃO E SEGURANÇA NO MODELO DISTRIBUÍDO DO INGRES..............55

5.4 DEFINIÇÃO DAS RESTRIÇÕES INTEGRIDADE DISTRIBUÍDAS NO INGRES..............56

5.5 CUMPRIMENTO DAS RESTRIÇÕES INTEGRIDADE DISTRIBUÍDAS NO INGRES. 59

CONCLUSÃO..............62

CAPITULO 6..............64

RESTRIÇÕES DE INTEGRIDADE DISTRIBUÍDAS NO ORACLE..............64

INTRODUÇÃO..............64

6.1 REPLICAÇÃO DE DADOS NO ORACLE 64

6.1.1 REPLICAÇÃO *MULTIMASTER* 65

6.1.2 REPLICAÇÃO *SNAPSHOT* 67

6.1.3 REPLICAÇÃO HÍBRIDA 69

6.1.2 TRATAMENTO DE CONFLITOS DA REPLICAÇÃO NO ORACLE 69

6.2 DISTRIBUIÇÃO DE DADOS NO ORACLE 71

6.2.1 BANCOS DE DADOS DISTRIBUÍDOS HOMOGÊNEOS ORACLE 71

6.2.2 BANCOS DE DADOS DISTRIBUÍDOS HETEROGÊNEOS ORACLE 72

6.2.3 ADMINISTRAÇÃO E SEGURANÇA DOS BANCOS DE DADOS DISTRIBUÍDOS ORACLE 76

6.3 DEFINIÇÃO DAS RESTRIÇÕES DE INTEGRIDADE NO ORACLE 77

6.4 CUMPRIMENTO DAS RESTRIÇÕES DE INTEGRIDADE NO ORACLE. 80

CONCLUSÃO 83

CAPITULO 7 84

RESTRIÇÕES DE INTEGRIDADE DISTRIBUÍDAS NO DB2 84

INTRODUÇÃO 84

7.1 REPLICAÇÃO DE DADOS NO DB2. 84

7.1.1 CONTROLES DE REPLICAÇÃO DO DB2 86

7.1.2 MÉTODOS DE REPLICAÇÕES ABORDADAS PELO DB2 89

7.1.3 CENÁRIO DE REPLICAÇÃO SUPORTADAS PELO DB2 91

7.1.4 RESOLUÇÃO DE CONFLITOS NO DB2 94

7.1.5 RESTRIÇÕES DE REPLICAÇÃO NO DB2 94

7.2 DISTRIBUIÇÃO DE DADOS NO DB2 95

7.2.1 *REMOTE UNIT OF WORK* 95

7.2.1.1 *DISTRIBUTED REQUEST* 96

7.2.2.1 ATUALIZANDO APENAS UM BANCO DE DADOS 97

7.2.2 *DISTRIBUTED UNIT OF WORK* - DUOW 97

7.3 COMUNICAÇÃO E CONECTIVIDADE DO DB2 AO CONTEXTO DISTRIBUÍDO 99

7.4 DEFINIÇÕES DAS RESTRIÇÕES DE INTEGRIDADE DISTRIBUÍDAS NO DB2 101

7.5 CUMPRIMENTO DAS RESTRIÇÕES DE INTEGRIDADE DISTRIBUÍDAS DB2 102

CONCLUSÃO 105

CAPITULO 8 107

RESTRIÇÕES DE INTEGRIDADE DISTRIBUÍDAS NO SQL 2000 107

INTRODUÇÃO 107

8.1 REPLICAÇÃO DE DADOS NO SQL 2000 107

8.1.1 *SNAPSHOT REPLICATION* 108

8.1.2 *TRANSACIONAL REPLICATION.* 111

8.1.3 *MERGE REPLICATION* 113

8.1.4 RESOLUÇÃO DE CONFLITOS E SINCRONIZAÇÃO NO SQL 2000 116

8.1.5 OPÇÕES DE REPLICAÇÃO NO SQL 2000 117

8.1.6 COMUNICAÇÃO NA REPLICAÇÃO NO SQL 2000 118

8.2 DISTRIBUIÇÃO DE DADOS NO SQL 2000 119

8.2.1 TRANSAÇÕES DISTRIBUÍDAS 119

8.2.2 CONSULTAS DISTRIBUÍDAS 122

8.2.3 JUNÇÃO DA CONSULTA DISTRIBUÍDA E TRANSAÇÃO DISTRIBUÍDA . 125

8.2.4 COMUNICAÇÃO E SEGURANÇA NA DISTRIBUIÇÃO DE DADOS 126

8.1.5 RESTRIÇÕES NO *TRANSACT-SQL* EM *DISTRIBUTED QUERY* 128

8.3 DEFINIÇÕES DAS RESTRIÇÕES DE INTEGRIDADE DISTRIBUÍDAS 129

8.4 CUMPRIMENTO DAS RESTRIÇÕES DE INTEGRIDADE DISTRIBUÍDAS 130

CONCLUSÃO 134

CAPITULO 9 134

COMPARAÇÕES DOS SGBD'S PARA O MODELO DISTRIBUÍDO 134

INTRODUÇÃO 134

9.1 AVALIAÇÃO GERAL 134

9.2 COMPARAÇÃO NA REPLICAÇÃO DE DADOS 135

9.3 COMPARAÇÃO NA DISTRIBUIÇÃO DE DADOS 138

CONCLUSÕES FINAIS 140

REFERÊNCIAS BIBLIOGRÁFICAS 144

APÊNDICE A 151

Lista de Figuras

Figura 1 - Uma arquitetura Possível de Bancos de Dados Distribuídos 8
Figura 2 - Separação dos Passos do Projeto Distribuído 10
Figura 3 - Relação Curso Fragmentada Horizontalmente 11
Figura 4 - Relação Exame Fragmentada Verticalmente 11
Figura 5 - Fragmentação Híbrida 12
Figura 6 - Camadas de Transparências 13
Figura 7 - Relacionamento entre os Principais Problemas do Ambiente Distribuído 21
Figura 8 - Banco de Dados com Informações Incorretas 26
Figura 9 - Modelo de Entidade Relacionamento 27
Figura 10 - Exemplo de Especialização/Generalização 29
Figura 11 - Arquitetura do *Distributed Constraint Systems* - DCMS 43
Figura 12 - Arquitetura Simples de Replicação do Ingres 48
Figura 13 - Arquitetura de Distribuição de Dados no Ingres 54
Figura 14 - Replicação *Multimaster* 66
Figura 15 - Replicação *Snapshot Read-Only* 67
Figura 16 - Replicação *Snapshor Updateable* 69
Figura 17 - Exemplo de Sistema de Bancos de Dados Distribuídos Homogêneos 72
Figura 18 - Domínio de Rede e Nomes Globais dos BD's Oracle 74
Figura 19 - Replicação do Tipo *Data Distribution* do DB2 91
Figura 20 - Replicação do Tipo *Data Consolidation* do DB2 92
Figura 21 - Replicação do Tipo *Update-anywhere* com Risco de Conflito 92
Figura 22 - Replicação do Tipo *Update-anywhere* sem Risco de Conflito 93
Figura 23 - Replicação do Tipo *Occasionally Connected* 93
Figura 24 - Figura 24: *Unit of Work* em um Simples Banco de Dados DB2 96
Figura 25 - Usando Múltiplos Bancos de Dados em uma Simples Transação 97
Figura 26 - Atualizando Múltiplos Bancos de Dados em uma Simples Transação 98
Figura 27 - Troca de Dados entre Estação e Servidor de DRDA 100
Figura 28 - Arquitetura do *Snapshot Replication* 111
Figura 29 - Arquitetura da *Transactional Replication* 112

Figura 30 - Arquitetura da *Merge Replication* 114
Figura 31 - Transação Distribuída com Coordenação Própria no MS DTC 121
Figura 32 - Aplicações de SQL 2000 Invocando Diretamente Transação Distribuída 122
Figura 33 - Estrutura Básica do Acesso à Dados Pela Consulta Distribuída 125
Figura 34 - Processamento de Regras em Relação aos Estados do Banco de Dados 159
Figura 35 - Modelo de Criação e Funcionamento de *ECA-rule*. 160
Figura 36 - Arquitetura do Sistema de Regras Ativas 164
Figura 37 - Arquitetura do *ECA-rule* Distribuída 168
Figura 38 - Grafo de Dependência de Disparo de Regras 172
Figura 39 - Grafo de Confluência do Comportamento de Regras 173

Lista de Tabelas

Tabela 1 – Opções de Replicações no SQL 2000 117
Tabela 2 - Comparação entre os SGBD's para o Modelo Distribuído de Dados ... 135
Tabela 3 – Características Essenciais na Replicação de Dados............................. 136
Tabela 4 – Opções de Replicações e Restrições de Integridade............................. 136
Tabela 5 – Dimensão do Modelo de Regras Ativas.. 154
Tabela 6 - Dimensão do funcionamento do *ECA-rule*... 161

Lista de Abreviaturas

2PC	*Two-phase commit*
ADO	*Active Data Object*
AIX	*Advanced Interactive eXecutive*
ANSI	*American National Standart Institute*
API	*Application Program Interface*
BDD	*Banco de Dados Distribuído*
CAI	*Computer Associates Inc.*
CDB	*Coordinator DataBase*
CDDS	*Consistent Distributed Data Set*
CMW	*Compartmented Mode Workstations*
CODASYL	*Conference on Data Systems Language*
DBA	*DataBase Administrator*
DCMS	*Distributed Constraint Management System*
DDL	*Data Definition Language*
DDMA	*Distributed Data Management Architecture*
DEC	*Digital Equipment Corporation*
DER	*Diagrama de Entidade Relacionamentos*
DES	*Data Encryption Standart*
DJRA	*DataJoiner Replication Administration*
DLL	*Dynamic-link Library*
DMCM	*Distributed Multi-cache Management*
DML	*Data Manipulation Language*
DRDA	*Distributed Relational Database Architecture*
DTP	*Distributed Transaction Processing*
DUOW	*Distributed Unit of Work*
E-C-A	*Evento-Condição-Ação*
ER/E	*Entidade Relacionamento/Estendida*
FD:OCA	*Formatted Data Object Content Architecture*
FTP	*File Transfer Protocol*
GCM	*Global Constraint Manager*
IBM	*International Busines Machine*
LCM	*Local Constraint Manager*

LOB	*Larger Object Binary*
MER	*Modelo Entidade Relacionamento*
MS-DTC	*Microsoft Distributed Transaction Coordinator*
MTS	*Microsoft Transaction Server*
NASA	*National Aeronautics Space Agency*
NCSC	*National Communications Security Evaluation*
ODBC	*Open Database Conectivity*
OLE DB	*Object Linking And Embedding Database*
PL/SQL	*Procedural Language/Structure Query Language*
SGBD	*Sistema Gerenciador de Banco de Dados*
SGBDD	*Sistema Gerenciador de Banco de Dados Distribuído*
SGBDR	*Sistema Gerenciador de Banco de Dados Relacional*
SH	*Serviço de Heterogeneidade*
SNA	*System Network Architecture;*
SPX/IPX	*Sequenced Packet Exchange/Internetwork Packet Exchange*
SQL	*Structure Query Language*
SSL	*Security Sockets Layer*
TCP/IP	*Transmission Control Protocol*
TM	*Transaction Monitor*
UOW	*Unit of Work*
X/Open XA	*X/Open eXtended Architecture*
X/Open DPT	*X/Open Distributed Processing Transaction*

Resumo

Palavras-chave: bancos de dados distribuídos, restrições de integridade, regras ativas, replicação de dados, distribuição de dados, *ECA-Rule*.

Esta dissertação ressalta e analisa uma das possibilidades de aplicação da tecnologia de banco de dados (que é o ambiente distribuído), onde várias instâncias de bancos de dados são interligadas por redes de computadores independentes da localização geográfica, constituindo uma integração transparente e autônoma. Porém, esse ambiente apresenta vários problemas, dentre eles destacamos as restrições de integridade distribuídas, sob os aspectos de definição e de cumprimento das restrições de integridade. Para tanto, admitimos os principais sistemas gerenciadores de banco de dados disponíveis no mercado atualmente (Ingres II, Oracle8i, DB2 v7 e SQL 2000), nos quais analisamos as formas que os mesmos suportam a replicação e distribuição de dados. Ciente das propriedades de replicação e distribuição de dados de cada SGBD, verificamos a maneira pela qual as regras de integridade são concebidas e mantidas por eles. Neste sentido, avaliamos cada SGBD (considerando o domínio de aplicação) em conformidade às respectivas características. Outro modelo investigado neste trabalho, que pode ser adotado como um forte aliado ao cumprimento das restrições de integridade distribuídas, é o paradigma de regras ativas *(ECA-Rule)*.

Abstract

Key words: distributed databases, distributed integrity constraints, actives rules, data replication, data distribution, *ECA-Rule*.

This paper points out and analyses one of the possibilities of using the database technology (which is the distributed environment), where several database ways are linked by computer networks irrespective of geographic positions, constituting a clear and autonomous integration. Howener this environment presents several problems, among them we highlight the distributed integrity restrictions considering the aspects of definition and fulfillment of the integrity restrictions. For that, we accept the main management systems of the databases available on the market nomadays, (Ingres II, Oracle8i, DB2 v7 e SQL Server 2000), in which we analyse the ways DBMS hold the data replication and distribution. Knowing the data distribution and replication proprieties for each DBMS, we have studied the forms by which rules for integrity are established and kept by them. In this meaning, we evaluated each DBMS (considering their application control) in according to the respective characteristics. Another model investigated in this paper, that may be as a strong allied to the fulfillment of the distributed integrity restrictions, is the active paradigmatic rules (ECA-Rule).

INTRODUÇÃO

Um banco de dados consiste, conforme [ELM 2000], em uma coleção de dados relacionados que representam alguns aspectos do mundo real. E o sistema gerenciador do banco de dados, é o software projetado para assistir a manutenção e utilização dos mesmos [RAM 2000].

Historicamente, a tecnologia de banco de dados surgiu nos anos 50 com os sistemas de arquivos de dados simples, sendo os mesmos acessados seqüencialmente. Mas a idéia de integração entre a base de dados e softwares de gerenciamento aconteceu no início dos anos 60, desenvolvida por Charles Bachman na *General Electric,* e denominado *Integrated Data Store*. Essa idéia serviu como base para o desenvolvimento do modelo *network data model* (modelo em rede), padronizado pela *Conference on Data Systems Languages* (CODASYL). Mais tarde, a IBM - *International Busines Machine* desenvolveu o *Information Management System* (IMS) precursor do *Database Management System* (DBMS) usado hoje na maioria das instalações de banco de dados. O IMS serviu como alternativa para a representação de um *framework* de dados, chamado *hierarquical data model* (modelo hierárquico). Nos anos 70, também na IBM, Edgard Codd propôs uma nova representação chamando-a de *relational data base* (modelo relacional) [RAM 2000]. Desde então, a popularidade do modelo relacional consolidou-se nas organizações, devido aos avanços na tecnologia de microprocessadores e de redes computadores locais, permitindo a troca de informações em parcelas de tempo muito pequenas.

A evolução da tecnologia de banco de dados, tem acompanhado quase sempre as necessidades das organizações em obter informações, manipulando dados oriundos de diversas fontes. De acordo com a demanda por informações, a tecnologia de banco de dados é uma ferramenta indispensável ao contexto organizacional. Hoje, a tecnologia de banco de dados possui extensões para suportar diferentes domínios de aplicações como: gerenciamento paralelo e distribuído de banco de dados; banco de dados na internet; *data warehousing* e complexas consultas para suporte à decisão; mineração de dados;

banco de dados orientado a objetos; gerenciamento de banco de dados espacial; gerenciamento de banco de dados orientado a regras; dentre outras.

Das extensões de banco de dados mencionadas acima, duas delas serão tratadas neste trabalho: bancos de dados distribuídos e regras ativas.

Em bancos de dados distribuídos, os dados são armazenados em vários *sites,sendo que* cada *site,* é gerenciado por um sistema gerenciador de banco de dados que possui autonomia própria, ou seja, ele pode executar operações locais independentemente dos outros *sites.* A despeito da existência de várias possibilidades em distribuir dados, a visão clássica é que o contexto distribuído suporte duas propriedades: a independência dos dados distribuídos e a atomicidade das transações distribuídas. Dessa maneira, para o usuário do sistema de bancos de dados distribuídos, as operações são transparentes, pois ele não percebe quando e onde elas ocorrem.

Dentre as várias possibilidades do uso de aplicações de bancos de dados distribuídos, tomamos o exemplo de uma universidade multi-campi distribuída em regiões geograficamente distintas, onde cada campus possui sua autonomia e seus próprios dados. Todavia, a integração dos dados se faz necessária, principalmente para a administração financeira e acadêmica dessa universidade. Quanto à integridade desses dados, os sistemas gerenciadores deverão cumprir as restrições, tanto local quanto global, ou seja, se um aluno efetua o pagamento de sua mensalidade na tesouraria de seu campus, nesse momento a baixa desse pagamento deve refletir globalmente, pois eventualmente outra pessoa pode querer efetuar o mesmo pagamento em outro campus. Este é um simples exemplo, o qual deve ser observado para um sistema distribuído de banco de dados acadêmico.

Os bancos de dados que suportam regras ativas (basicamente *triggers),* além de efetuarem o gerenciamento de dados também executam ações em resposta a eventos, como as alterações dos próprios dados. Regras ativas especificam quando e quais ações deverão ser executadas. O modelo de regras ativas (evento-condição-ação) é

amplamente usado, podendo ser adotado para várias finalidades inclusive no cumprimento da restrição de integridade [SIL 2000].

Objetivo Geral

O principal objetivo deste trabalho é analisar e comparar restrições de integridade em ambientes de bancos de dados distribuídos, além de uma discussão relacionada à funcionalidade e aplicabilidade das regras ativas.

Nessa análise comparativa serão abordados quatro dos principais sistemas gerenciadores de bancos de dados - SGBD (Ingres, Oracle, DB2 e SQL 2000), objetivando verificar a definição e o cumprimento das restrições de integridade para o contexto distribuído. E ainda, comparar como cada sistema gerenciador de banco de dados implementa a distribuição e a replicação de dados.

Objetivos Específicos

Os objetivos específicos são alicerçados pela literatura de banco de dados, assim como pela documentação técnica de cada fabricante dos sistemas gerenciadores de banco de dados envolvidos neste trabalho:

- Apresentar o contexto distribuído com suas vantagens e desvantagens;
- Esclarecer o conceito de restrição de integridade, sob a forma de definição e cumprimento da mesma, independentemente do contexto do banco de dados (centralizado ou distribuído);
- Contextualizar a restrição de integridade ao ambiente distribuído de banco de dados;
- Analisar a documentação técnica de cada SGBD (Ingres II, Oracle8i, DB2 e SQL Server 2000) e apresentar os mecanismos de replicação e distribuição de

dados, bem como a definição e o cumprimento da restrição de integridade para esses mecanismos;

- Demonstrar a forma que cada SGBD trata a restrição de integridade distribuída;
- Efetuar comparações entre SGBD's em relação à definição e o cumprimento das restrições de integridade distribuídas, bem como o suporte de distribuição e replicação de dados;
- Apontar as características de cada SGBD que se destacam em relação às outras possibilidades de aplicação apresentadas neste trabalho;
- Demonstrar a aplicabilidade das regras ativas, suas dimensões, funcionamento e estrutura;
- Contextualizar as regras ativas nos modelos: relacional e orientado a objetos;
- Focalizar regras ativas para o ambiente distribuído;
- Descrever sucintamente o desenvolvimento de aplicações ativas para banco de dados.

Justificativa

Atualmente, a tecnologia da informação envolve diretamente ou indiretamente todas as camadas da sociedade, sobretudo as organizações que podem agregar valores às suas atividades fazendo uso dessa tecnologia. Entretanto, a adoção dessa tecnologia requer conhecimentos do ambiente no qual será inserida, bem como dos artefatos envolvidos (*hardware*, *software* e telecomunicação). Seguindo esse contexto, o presente trabalho envolve duas tecnologias: banco de dados e redes de computadores. Ambas proporcionam a troca de informações em parcelas pequenas de tempo, tornando cada vez mais ágil a disseminação da informação na organização e/ou entre organizações.

Bancos de dados distribuídos permitem a integração de dados alocados em *sites* de um sistema distribuído, cada um com autonomia própria. Dentre as possibilidades de distribuição e replicação de dados, a principal preocupação é com a integridade dos mesmos. Normalmente, cada base de dados distribuída compõe um esquema global, e se

uma dessas bases não mantém a integridade de seus dados, então todo o ambiente estará comprometido. Cada nova instância de qualquer banco de dados que forma o ambiente distribuído, deverá possuir consistência compatível aos demais. Daí a necessidade de verificarmos como os SGBD's tratam essa questão, considerando a definição e o cumprimento das restrições de integridade. O que requer atenção para a forma como cada um deles implementa a distribuição e/ou replicação de dados. Nessa verificação os principais sistemas gerenciadores de banco de dados profissionais (Ingres, Oracle, DB2 e SQL 2000) serão objetos de estudo.

O modelo de regras ativas é uma extensão da tecnologia de banco de dados e possui várias aplicabilidades, dentre elas ressalta-se a definição e o cumprimento das restrições de integridade, tanto em ambiente centralizado como distribuído.

Estrutura

O trabalho está estruturado de forma que paulatinamente se obtenha os conceitos necessários à compreensão do domínio tratado, e à verificação de que o assunto é pouco investigado, principalmente na questão prática.

Assim sendo, o capítulo 2 contribuirá para o esclarecimento sobre bancos de dados distribuídos e suas aplicações, bem como os problemas encontrados para esse contexto como: projeto distribuído, gerenciamento de transações distribuídas, controle concorrência, dentre outros.

O capítulo 3 abordará as restrições de integridade sob duas perspectivas: de definição e de cumprimento dessas restrições. No cumprimento das restrições também se destacam as verificações pessimista e otimista.

No capítulo 4, serão tratadas as características das restrições de integridade para bancos de dados distribuídos, com relevância aos algoritmos que procuram otimizar a

verificação da integridade dos dados distribuídos, através de uma consulta local, onde cada consulta compõe um esquema global das restrições de integridade.

No capítulo 5, será verificada, a forma pela qual o SGBD Ingres II realiza a replicação e distribuição de dados. Como ponto fundamental deste trabalho, serão investigados a definição e o cumprimento das restrições de integridade para o ambiente distribuído suportado pelo SGBD Ingres II.

Nos capítulos 6, 7 e 8, serão discutidas as mesmas questões descritas no capítulo anterior, mas direcionadas para os respectivos SGBDs: Oracle8i, DB2 v7 e SQL *Server* 2000 .

No capítulo 9, serão apresentados os resultados comparativos das análises realizadas nos SGBDs (conforme capítulos 5,6,7 e 8), sobretudo as questões de definição e cumprimento das restrições de integridade e suporte às fragmentações horizontais e verticais.

Finalmente, no capítulo 10 serão traçados comentários finais e propostas para posteriores trabalhos.

No apêndice A descreve-se o modelo de regras ativas, o qual permite excluir a passividade normalmente encontrada nos bancos de dados quando implementado convencionalmente. Nessa descrição, envolve-se as dimensões do *eca-rule* (evento-condição-ação), bem como as características estruturais, comportamentais e o desenvolvimento das regras ativas. Mas a principal preocupação é a contextualização das regras ativas ao cumprimento das restrições de integridade distribuidas.

Capítulo 2

Bancos de Dados Distribuídos

Bancos de Dados Distribuídos (BDD) é uma tecnologia emergente implementada pelos Sistemas Gerenciadores de Banco de Dados (SGBD), com módulos de suporte ao gerenciamento e distribuição de dados. O esse ambiente é suportado por duas tecnologias: tecnologia de banco de dados e tecnologia de rede de computadores (comunicação de dados). Com o acesso facilitado a essas tecnologias, as organizações têm muito interesse em descentralizar seus processos com a finalidade de integrar, e ao mesmo tempo manter as características das informações de diversas origens, contidas em diferentes regiões geográficas.

Ao contrário dos bancos de dados centralizados, no qual os processadores são fortemente acoplados e constituem um único sistema de banco de dados, bancos de dados distribuídos possuem *sites* que são fracamente acoplados compartilhando componentes físicos. Além disso, os sistemas gerenciadores de banco de dados de cada *site* podem possuir um alto grau de independência mútua. Cada *site* pode participar da execução de uma transação que acessa um dado em um ou diversos *sites* [SIB 1999].

Um banco de dados distribuído é uma coleção de bases de dados inter-relacionadas e distribuídas em *sites* ou nós sob uma rede de computadores, e o sistema gerenciador de banco de dados distribuído tem como principal tarefa, gerenciar essas bases de forma transparente ao usuário de um sistema de bancos de dados distribuídos [ELM 2000] [OZS 1999].

O funcionamento de um banco de dados distribuído envolve um conjunto de mecanismos de controle, tornando a implementação desse tipo mais arrojada e complexa. O objetivo desse capítulo é elucidar o contexto de banco de dados distribuído dentre outras possibilidades de implementação da distribuição de acesso aos dados (arquitetura cliente/servidor, múltiplos-clientes/múltiplos-servidores, sistema de banco

de dados federados e *World Wide Web* banco de dados), abordando características como: aplicabilidade; distribuição de dados e transparência. Além disso, é indispensável à contextualização dos principais problemas encontrados nesse ambiente. Uma sugestão de arquitetura de um banco de dados distribuído está disposta na figura 1, esta apresenta *sites* interconectados através de uma rede de computadores com as respectivas bases de dados autônomas.

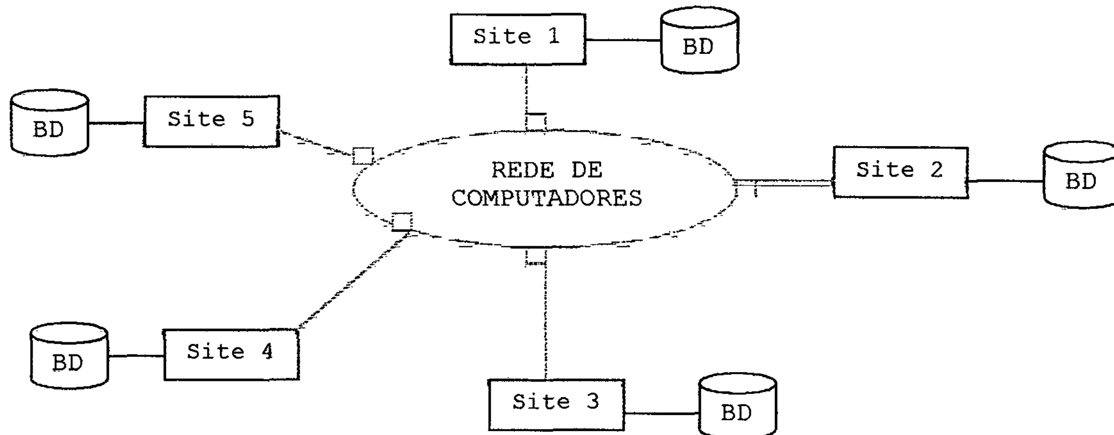

Figura 1: Uma Arquitetura possível de Bancos de Dados Distribuídos.

2.1 Aplicabilidade

Com a globalização econômica, torna-se cada vez mais comum a aquisição,fusão ou incorporação entre organizações. Fato que desperta para necessidade de obtenção de informações oriundas de diversas repartições organizacionais, mas não descaracterizando o ambiente e autonomia local. Então os projetistas de banco de dados, muitas vezes decidem em manter a estrutura de dados (projeto *botton-up*) existente (com algumas adaptações), em outros casos concebem novas bases de dados (projeto *top-down*). Em ambos os casos as bases de dados são interconectadas.

De forma geral, aplicação principal dos bancos de dados distribuídos é permitir às organizações a descentralização dos processos de negócios, e ao mesmo tempo acessá-los transparentemente. Isso preserva a característica e autonomia de cada base de dados.

Os bancos de dados distribuídos oferecem às organizações mais flexibilidade e modularidade na forma como os bancos de dados são organizados e usados.

Em bancos de dados distribuídos não se tem aplicabilidade específica, mas sim um conjunto de possibilidades. Essas possibilidades são implementadas em detrimento ao domínio de problema tratado. Por exemplo:

- Uma organização que possui vários escritórios, cada um deles, por sua vez pode criar, gerenciar e usar seus próprios bancos de dados, e as pessoas de outros escritórios podem acessar e compartilhar esses dados.

- Em outra situação, a organização possui centros específicos de negócios como: centro de produção; centro de pesquisa e desenvolvimento; centro de distribuição; centro administrativo; e, pequenos postos de vendas. Cada centro possui sua própria base de dados contento informações relevantes ao seu domínio de negócio, mas quando for necessário obter informações de outro(s) centro(s), o sistema gerenciador de banco de dados distribuído deve fornecer o acesso e manipulação desses dados.

2.2 Distribuição de Dados

A estrutura e a acomodação dos dados é fortemente ligada ao projeto de banco de dados, que tradicionalmente se divide em três partes: projeto conceitual, que representa em alto nível a realidade do domínio do problema (independente do SGBD); projeto lógico, traduz esta representação em estrutura de dados conforme vista pelo usuário do SGBD (dependente do SGBD) e projeto físico, que determina a estrutura de armazenamento físico e os mecanismos de manipulação.

No projeto de banco de dados distribuído, tradicionalmente, começa-se a tratar da fragmentação de um esquema global e alocação dos dados fragmentados a partir do nível lógico, descartando o nível conceitual [MES 1998].

Fragmentação é uma técnica eficiente na organização de dados aplicada ao contexto distribuído, pois não basta apenas distribuir relações é necessário que se tenha bom senso em projetar e alocar um banco de dados distribuído. Essa tarefa pode ser desenvolvida passo a passo (figura 02). Na aplicação da fragmentação dispomos de três estratégias: fragmentação horizontal, fragmentação vertical e fragmentação híbrida.

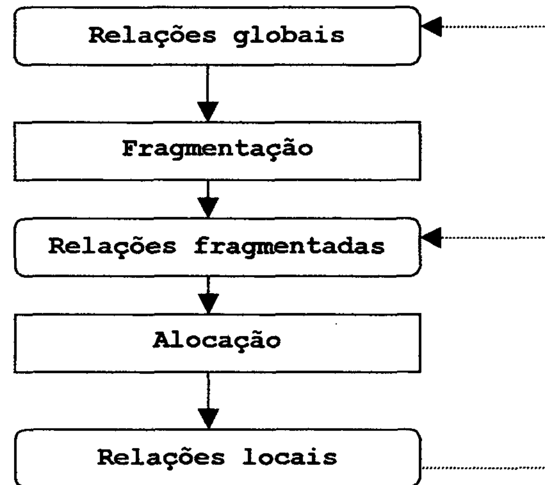

Figura 02: Separação dos Passos do Projeto Distribuído.

Faz-se necessário a corretude de uma fragmentação aplicada ao modelo inicial (MER – Modelo Entidade Relacionamento). Para tanto dispomos de três propriedades: *completude*, cada elemento da instanciação do DER (Diagrama Entidade Relacionamento) original deve estar presente na instanciação de algum dos fragmentos; *reconstrução,* o modelo original deve ser reconstrutível a partir de fragmentos; e, *disjunção,* cada elemento da instanciação do DER original deve estar presente em apenas um fragmento. Esta condição é relaxada na fragmentação vertical [MES 1998].

A fragmentação horizontal (figura 3) particiona relação em suas tuplas. Portanto, cada fragmento possui um subconjunto de tuplas da relação fragmentada. Há duas versões de particionamento horizontal: primária e derivada. A fragmentação horizontal primária de uma relação é a execução de predicados sobre a própria relação. Por outro lado a fragmentação horizontal derivada é o particionamento de uma relação que resulta da definição de predicados de outra relação. Conforme a figura 3, a fragmentação da

relação *Curso* obedece ao predicado de restrição (*código* $\geq$ *3 and* $\leq$ *4*), onde os cursos relacionados à computação deverão estar em separados na relação *Curso2* (fragmento).

Curso

Código	Nome
01	Física
03	C. da Computação
04	Sist. de Informação

Curso1

Código	Nome
01	Física

Curso2

Código	Nome
03	C. da Computação
04	Sist. De Informação

Figura 03: Relação Curso Fragmentada Horizontalmente.

Ao contrário da fragmentação horizontal, a fragmentação vertical (figura 4) de uma relação *R* produz fragmentos R_1, R_2,...R_R e que cada um deles contém um subconjunto dos atributos de *R*, bem como a chave primária de *R*.

Curso

Estudante	Grade	Curso
937653	B	01
937685	C	03
200888	A	04

Exame1

Estudante	Grade
937653	B
937685	C
200888	A

Exame2

Estudante	Curso
937653	01
937685	03
200888	04

Figura 04: Relação Exame Fragmentada Verticalmente.

Na fragmentação vertical obtemos uma nova estrutura derivada da relação original *Exame*, agora a relação (fragmento) *Exame1* possui dados somente da grade do estudante, enquanto que *Exame2* apresenta o curso de cada aluno.

A Fragmentação Híbrida também chamada de *mixed* ou fragmentação aninhada, é usada quando uma simples fragmentação horizontal ou vertical não satisfaz as necessidades da aplicação, e nesse caso a fragmentação horizontal pode ser seguida pela fragmentação vertical, ou vice versa, mas sempre uma após a outra produzindo uma estrutura de três camadas [OZS 1999]. A figura 5 destaca a relação (R) sendo primeiramente fragmentada horizontalmente, e em seguida verticalmente.

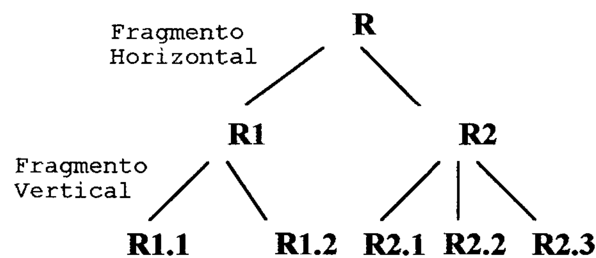

Figura 05: Fragmentação Híbrida.

Considerando qualquer nível de fragmentação, os quais deverão ser alocados em vários *sites* da rede. No entanto, deve se ter conhecimento preciso dos problemas que pode originar em função de uma alocação mal projetada, principalmente com o custo de atualização (inserção, deleção e recuperação) desses fragmentos. Além dos dados, também são relevantes a alocação dos algoritmos e as funções de controles envolvidas no projeto distribuído.

Na implementação da alocação encontramos três opções: particionado, onde cada fragmento é encontrado em um único lugar, mas cada *site* possui seu conjunto de fragmentos; replicado, ao contrario do anterior, todos os fragmentos são distribuídos em todos os *sites;* e, parcialmente replicado ou replicado e particionado. Alguns fragmentos podem ser replicados em um ou mais *sites,* assim como outros fragmentos podem ser encontrados em um único local.

2.3 Transparência

A transparência é a separação semântica do sistema em relação à implementação. Em outras palavras, esconde os detalhes da implementação do usuário, sendo esta uma

característica óbvia dos sistemas gerenciadores de banco de dados, seja distribuído ou centralizado. Quando um usuário dispara uma aplicação de consulta (*query* SQL – *Structure Query Language)*, a qual busca os dados em vários *sites,* eles são apresentados sem nenhuma pista de que houve acesso em várias bases de dados que normalmente estão distantes.

A transparência envolve camadas específicas que devem ser tratadas individualmente, porém a interdependência e a integração são fundamentais para que aplicação distribuída contemple o usuário. A figura 6 apresenta essas camadas, as quais serão descritas a seguir.

-> Camada de Linguagem
-> Camada de Fragmentação
-> Camada de Replicação
-> Camada de Rede
-> Camada de Independência de Dados

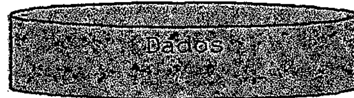

Figura 06: Camadas de Transparências.

A primeira camada, é a camada de independência de dados, que promove a independência entre as aplicações do usuário e a definição e organização do banco de dados. Ela ocorre em dois níveis: nível lógico e nível fisico. A independência lógica de dados refere-se a imunidade das aplicações usuárias em relação a estrutura lógica do banco de dados. Já independência fisica de dados, esconde os detalhes da estrutura de armazenagem dos dados das aplicações dos usuários [OZS 1999].

Na transparência da camada de rede, deve-se disponibilizar somente os recursos necessários ao usuário. No ambiente centralizado, essa disponibilidade acontece enquanto que no distribuído, além desse tipo de gerenciamento existem outros serviços de controles de acesso como transparência de localização, ou seja, independente da localização fisica do usuário ele pode executar uma tarefa que lhe é pertinente. Outra questão é a combinação da identificação com a localização do usuário. Se possível, não

deixar transparecer ao usuário que ele utiliza uma estação de trabalho conectada a uma rede de computadores.

A camada de transparência da replicação envolve principalmente a redundância da base de dados em vários *sites,* que sob o ponto de vista do usuário deve ser transparente, pois, se uma cópia que está sendo acessada falhar, automaticamente a aplicação pode redirecioná-la a outro *site*. Sob o ponto de vista da aplicação essa facilidade envolve problemas como atualização das bases, sincronismo, custo de acesso, e ainda reduz consideravelmente a independência de dados.

A camada de transparência de fragmentação tem o mesmo princípio da camada anterior, esconder os fragmentos dos usuários. Por outro lado, não se tem os mesmo efeitos oferecidos pela camada de replicação, pois o gerenciamento focaliza conjuntos de fragmentos. Para que isso ocorra, faz-se necessário o uso de estratégias na elaboração e na distribuição dos fragmentos. Quanto à recuperação, surgem *queries* (consultas) globais que também poderão ser fragmentadas.

A última camada trata da transparência de linguagem, que é bastante genérica e permite aos usuários um alto nível de acesso aos dados (linguagens de quarta geração, interface gráficas, mediadores, linguagens natural de acesso, entre outros...) [OZS 1999].

2.4 Problemas do Contexto Distribuído

A adoção do banco de dados distribuído pelas organizações concretiza a descentralização de seus negócios, proporcionando vantagens (descrita na seção 2.1) sobre o ambiente centralizado. Mas os fatores complicadores, aqueles encontrados no banco de dados centralizados, agora são multiplicados pelo número de *sites*, além de outros problemas caracterizados pelo sistema gerenciador de banco de dados distribuído.

Os problemas característicos do banco de dados distribuído são influenciados por três fatores principais: a) havendo replicação da base de dados, ou parte dela, todas as entradas e/ou atualizações executadas, devem refletir em cada elemento do dado replicado; b) se algum *site* falhar, ou *link* de comunicação for interrompido (tornando um ou mais *sites* incomunicáveis) enquanto uma atualização é executada, o sistema deve contornar os reflexos da falha até que o sistema esteja restabelecido; c) considerando que cada *site* não tem informação instantânea, ou as ações estão carregadas em outros *sites* (oposto do centralizado), então a sincronização das transações em múltiplos *sites* é considerada pesada [OZS 1999].

Esses fatores apontam para alguns problemas potencialmente complexos, que serão abordados a seguir:

2.4.1 Problemas no Projeto Distribuído

Ao contrário do projeto centralizado, um projeto de banco de dados distribuído envolve decisões em combinar as opções (relevante ao universo de discurso) de alocação (particionado, replicado ou parcial replicado), onde os esquemas dos bancos de dados devem representar a realidade semântica de cada *site*, acomodando cada fragmento, além disso, as aplicações são executadas contra os *sites*.

Fortemente ligado ao projeto estão os diretórios distribuídos, nos quais constam informações sobre os elementos de dados (localização e descrição). Um diretório pode ser uma entrada global para SGBDD ou local para cada *site;* pode ser centralizada para um *site* e distribuído para os demais, ter uma simples cópia ou múltiplas cópias. Há muitas pesquisas nessa área envolvendo programação matemática, para minimizar e combinar o custo de armazenagem do banco de dados, e também o processamento de transações e comunicação. Este é um problema de *NP-difícil* [LEW 1998], para o qual as soluções são baseadas em heurísticas [OZS 1999].

2.4.2 Gerenciamento de Transações Distribuídas

Esta seção destaca o problema de gerenciamento de transações distribuídas (não desconsiderando a relevância de cada um deles) pela forte relação entre o controle de concorrência, mecanismos de recuperação (tolerância a falhas), consistência dos dados e a global estrutura do sistema.

O gerenciamento de transações distribuídas requer procedimentos com vários outros problemas relacionados à confiabilidade, controle de concorrência, e utilização eficiente dos recursos envolvidos na execução da transação distribuída. Além disso, a própria característica da transação distribuída permite o processamento paralelo dentro de transações. O problema no gerenciamento de transações ocorre pelo fato da execução de transações em paralelo em múltiplos *sites.*

Controle de concorrência é a parte de manipulação de transações que negocia com múltiplos acessos os recursos compartilhados do sistema, para que esses acessos possam ser executados sem causar conflitos, permitindo, com isso, o compartilhamento dos recursos de forma transparente aos usuários [DEM 2001].

O gerenciamento de transações distribuídas se configura como uma tarefa bastante complexa e relevante, mas alguns mecanismos como: bloqueio em transações distribuídas, controle de concorrência otimista em transações distribuídas, e *timestamping,* podem tornar tal circunstância possível, ressaltando que há vantagens e desvantagens.

2.4.2.1 Bloqueio em Transações Distribuídas

Bloqueio *(locking)* precursor e amplamente difundido como um mecanismo de controle de concorrência. Esse mecanismo consiste no fato de que, quando algum processo (que inicia a transação) requer operações de leitura ou gravação sobre tuplas (ou objetos) como parte da transação, ele deve primeiro obter o bloqueio do recurso ou dos recursos que irá utilizar. Outros processos que requeiram tais recursos não irão obtê-

los, pelo fato destes estarem bloqueados. Existem dois tipos de bloqueios: o de leitura *read lock*, e o de gravação *write lock* [TAN 1992].

O problema dessa abordagem reside na possibilidade de uma transação ou várias transações requisitarem múltiplos bloqueios, visto que elas podem executar suas operações em diversos *sites*. As transações podem somente obter um bloqueio se elas não tiverem previamente liberado qualquer bloqueio. Isso implica que transações experimentam fases de requisições e fases de liberações para obterem e liberarem bloqueios. Todavia, isso pode levar a situação de *deadlock*[1], na qual uma transação esta esperando por um objeto bloqueado por uma outra transação, que em fila, está bloqueada esperando pelo objeto bloqueado pela primeira [REI 1997].

2.4.2.2 Controle de Concorrência Otimista em Transações Distribuídas

Outra maneira de manipular múltiplas transações concorrentes é o controle de concorrência otimista, o qual se vale de uma técnica bastante simples: tentar realizar atualizações, baseadas no fato de que elas, geralmente, não intervêm umas nas outras. Entretanto, conflitos não estão totalmente descartados, sendo necessário manter registros de objetos que estão sendo lidos ou escritos. No momento da atualização, ele verifica as demais transações na tentativa de averiguar se algum de seus objetos foi atualizado, desde que a transação foi iniciada. Em caso afirmativo, a transação é abortada, do contrário, a atualização é efetuada. A grande vantagem é a diminuição de *deadlock* [KUN 1981].

Em ambientes distribuídos, cada servidor envolvido necessita validar o acesso aos elementos de dados que ele manipula. Isto é realizado durante a primeira fase do protocolo *two-phase commit* [DEM 2001].

1: *Deadlock* é causado por duas ou mais transações dependentes uma(s) do final da(s) outra(s) para prosseguir a execução. A dependência de liberação de *locks* é mútua.

2.4.2.3 *Timestamping*

O *timestamping* destaca-se por ser um método diferente na implementação do controle de concorrência, o qual designa um valor (*timestamping*, ou identificador) para cada transação no momento em que ela ocorre. Através da utilização de algoritmos, como o de Lamport [LAM 1990], pode-se certificar de que estes identificadores são únicos, os quais correspondem à sua finalidade e princípios básicos de funcionamento. A idéia básica desta técnica é a seguinte [DAT 1988]:

- Cada transação recebe um identificador (*timestamp*) único global;
- As operações de atualizações não aplicadas fisicamente sobre o banco de dados até o término com sucesso da respectiva transação;
- Cada um dos objetos do banco de dados carrega o *timestamp* da transação que por último o leu (*read*), e o *timestamp* da transação que por último o atualizou (*write*);
- Se uma solicitação de operação sobre o banco de dados feita por determinada transação (T1), conflitar com alguma outra operação já executada, baseada na solicitação de outra transação mais jovem (T2), a transação (T1) é cancelada e reiniciada;
- Quando uma transação é reiniciada por algum motivo, esta receberá um novo identificador *timestamp*.

2.4.3 Execução de Consultas Distribuídas

A execução de consultas distribuídas também é um problema de *NP-difícil*, e sua abordagem é baseada em heurísticas, pois o problema reside em decidir uma estratégia para execução de cada consulta sobre a rede na maior distância de custo efetivo.

Nos sistemas gerenciadores de banco de dados distribuídos temos dificuldades em encontrar técnicas de otimização para processamento de consultas sobre a fragmentação e distribuição de dados. Na fragmentação a técnica usada é a consulta algébrica, a qual é especificada nas relações globais e transformada em fragmentos de operações. A

localização requer a otimização global das operações, as quais se responsabilizam por parte da otimização global da consulta. A otimização global da consulta envolve a permutação na ordem das consultas determinando a execução de *sites* para várias operações distribuídas, e identificando a melhor execução do algoritmo para operações distribuídas [OZS 1996].

Os fatores preponderantes são os custos de comunicação, e a falta de informação suficientemente disponível em cada site. O objetivo é abordar o paralelismo para melhorar a performance na execução das transações.

2.4.4 Controle de Concorrência

Controle de concorrência é um dos problemas mais estudados no contexto dos SGBDD's – Sistemas Gerenciadores de Bancos de Dados Distribuídos. O controle de concorrência envolve sincronização de acesso a bases de dados distribuídas, tendo como principal preocupação, manter a integridade dessas bases. Agora a preocupação não se restringe em apenas uma base de dados, mas em várias cópias de base de dados envolvendo consistência. Múltiplas cópias de elementos de dados convergem em valores como consistência mútua.

A transação é um dos principais focos desse problema e compreende duas situações básicas: a transação pessimista e a otimista. A pessimista sincroniza a transação requisitada antes da execução da mesma, e a transação otimista verifica-se há comprometimento da consistência do banco de dados no momento de sua execução[OZS 1999].

O grande desafio em sincronizar as transações executando o argumento de serializabilidade, é o controle de concorrência entre os algoritmos distribuídos, considerando que essas execuções são efetuadas em vários *sites* simultaneamente. Então serializabilidade global faz-se necessária e requer duas premissas: a) a execução de um

conjunto de transações em cada *site* é serializável; e b) a ordem da serialização dessas transações para todos os *sites* deve ser idêntica [OZS 1996].

Outra complexidade é quando os algoritmos baseiam-se em *locking,* o gerenciamento do *lock* pode ser centralizado ou distribuído, mas um amplo conhecimento de todos os efeitos do *lock* é indispensável, principalmente distribuído, e a concorrência entre eles pode causar *deadlocks.*

2.4.5 Confiabilidade

Ao apontarmos as facilidades do banco de dados distribuído (descrita na seção 2.1), envolvemos recursos adicionais para que isso se concretize, principalmente os recursos de rede. O funcionamento adequado do BDD depende da acessabilidade da estrutura de rede de computadores e também da operabilidade dos *sites.* Quando algum desses recursos está indisponível, mecanismos que asseguram a integridade das bases de dados devem ser acionados para que a detecção e recuperação desses recursos se estabeleçam o mais breve possível mantendo assim a confiabilidade.

2.4.6 Problemas Adicionais

Pode ocorrer que um ou mais *sites* possua base de dados heterogênea, remanescente de uma base autônoma e/ou centraliza existente, então ela deve ser tratada de forma canônica para facilitar a translação de dados na manipulação dos mesmos. Há também a preocupação com o suporte aos sistemas operacionais que envolvem várias camadas de redes, gerenciamento do SGBDD de cada *site,* largura de banda do *link* de comunicação, bem como a estabilidade da comunicação, e ainda o poder de processamento e a robustez do *hardware* envolvido no ambiente.

2.4.7 Relação Entre os Principais Problemas

Os problemas encontrados não são isolados, eles se relacionam, e muitas vezes um problema pode desencadear outros; ou ainda um problema novo pode surgir em detrimento da solução encontrada para o problema existente. O epicentro da relação entre os problemas reside no projeto do banco de dados, ele determina o grau de complexidade abordada. Gerenciamento de transações distribuídas, processamento de consultas, e. controle de concorrência são problemas que envolvem a manipulação dos dados distribuídos, portanto indicam também um alto grau de relacionamento e demanda complexidade. A figura 7 destaca o comprometimento entre os problemas encontrados para o ambiente de banco de dados distribuído.

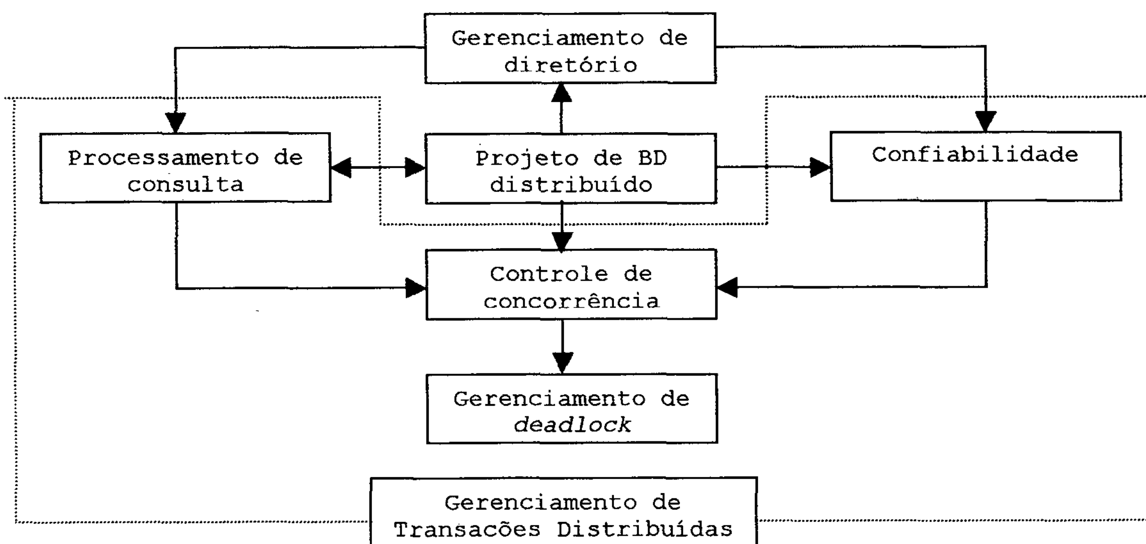

Figura 07: Relacionamento Entre os Principais Problemas do Ambiente Distribuído.

Considerações do Ambiente Distribuído

Como vimos neste capítulo o modelo distribuído para banco de dados possui vantagens e desvantagens e apresenta vários problemas. Dentre eles o principal deles é o gerenciamento de transações distribuídas, o qual possui fortes ligações com outros problemas (figura 07 envolvidos pela linha pontilhada). Entretanto algumas pesquisas têm proporcionado o uso desse modelo pelas organizações quando o ambiente é

favorável a ele, mas ainda há muito para se fazer em relação aos principais problemas que aparecem no modelo distribuído. Problemas esses que no modelo centralizado é mais simples de solucioná-los ou as vezes não aparecem.

Ao adotarmos um ambiente distribuído para banco de dados, devemos ter ciência dos obstáculos que poderão surgir, e ao mesmo tempo contorná-los. É imprescindível fazer uso do custo benefício ao optarmos pelo modelo de bancos de dados distribuídos.

Na parte que se segue serão tratados as questões de restrições de integridade como uma propriedade dos sistemas gerenciadores de banco de dados, independente do modelo adotado (distribuído ou centralizado).

Capítulo 3

Restrições de Integridade

Um banco de dados é útil somente quando há informações contidas, mas ao contrário, se as informações armazenadas estiverem incorretas, então as conseqüências serão catastróficas. Entretanto o SGBD deve prevenir a entrada de informações indesejáveis no banco de dados.

A restrição de integridade é uma condição especificada no esquema do banco de dados, com o objetivo de restringir a entrada de dados não desejáveis a uma instância do banco de dados. Se qualquer instância do banco de dados satisfaz todas as restrições especificadas no esquema do banco de dados, ela é considerada uma instância consistente. Qualquer operação (inserção, deleção e atualização) pode levar o estado do banco de dados consistente (atual) a um novo estado inconsistente, quando o SGBD não cumprir qualquer das restrições de integridade definidas no esquema do banco de dados.

Neste capítulo apresentamos alguns tipos de restrições de integridade, tanto no âmbito de definição, quanto no cumprimento das mesmas, desconsiderando o ambiente (centralizado ou distribuído).

Manter a consistência de um banco de dados envolve várias funções do SGBD, como: controle de concorrência; controle de segurança; controle de proteção; e, controle de integridade semântica. Controle de integridade semântica assegura a consistência do banco de dados por rejeitar aplicações que podem conduzir o banco de dados a um estado inconsistente, ou por ativar ações específicas que compensa os efeitos causados por uma aplicação que levou o banco de dados ao estado inconsistente [OZS 1999].

As restrições de integridade fornecem a garantia que as mudanças feitas no banco de dados por usuários ou aplicações autorizadas não resultem na perda da consistência dos

dados [SIL 1999]. Assim, as restrições de integridade são regras que representam o conhecimento e propriedades do domínio da aplicação para o modelo de dados. As regras de integridade possuem uma forte conexão com o modelo de dados, ou seja, quanto maior for a semântica da informação, maior será a proximidade dessas regras com a aplicação.

Segundo [OZS 1999], a restrição estrutural e a restrição comportamental compreendem os principais tipos de restrição de integridade. A restrição estrutural expressa basicamente a propriedade semântica inerente ao modelo do banco de dados. A restrição comportamental por outro lado, regulamenta o comportamento das aplicações. Cada tipo de restrição de integridade é essencial ao projeto de banco de dados independente de sua característica, pois evidenciam as propriedades, estruturas, dependências e associações entre os objetos participantes do banco de dados.

3.1 Definições das Restrições de Integridade Passiva

Na abordagem relacional costuma-se classificar as restrições de integridade nas seguintes categorias (previamente definidas na construção do modelo lógico de dados): integridade de domínio; integridade de vazio; integridade de chave e integridade referencial.

- Integridade de domínio: restrições deste tipo especificam que valor de um campo deve obedecer à definição de valores admitidos para uma coluna de cada relação (inteiro, real, alfanumérico de tamanho definido, etc.). Porém, o usuário do SGBD não pode definir domínios próprios;
- Integridade de vazio: este tipo de integridade especifica se os atributos de uma relação podem conter valores vazios ou não;
- Integridade de chave: trata-se da restrição que define a unicidade dos valores da chave primária e alternativa;

- Integridade referencial: este tipo de restrição define que os valores dos campos que aparecem em uma chave estrangeira devem aparecer na chave primária da tabela referenciada.

As restrições de integridade acima são garantidas automaticamente pelo SGBD relacional, ou seja, não é necessário nenhum procedimento adicional. Porém há muitas outras restrições de integridade que não se encaixam nas categorias acima e que normalmente não são garantidas pacificamente pelo SGBD. Essas restrições são chamadas de restrições semânticas (tratadas na seção 3.4) [HEU 1998].

As definições dessas restrições acontecem na concepção da relação, ou seja, diretamente nos comandos `CREATE TABLE` e `ALTER TABLE`, ou na interface de gerencia do banco de dados. Essas operações normalmente são efetuadas pelo DBA – *Data Base Administrator*.

```
CREATE TABLE [schema] Nome.Tabela (.....);
CREATE TABLE Aluno (
  RA          Number  (6) PRIMARY KEY,
  Nome        Varchar2 (20) NOT NULL,
  Sobnome     Varchar2 (30),
  DataAniv    Date,
  CodCurso    Number (4) NOT NULL,
  FOREIGN KEY (CodCurso) REFERENCES Curso (CodCurso);
```

3.1.1 Violações da Integridade Passiva

A violação da restrição de integridade proporciona o estado incorreto do banco de dados. Por outro lado, não são apenas as violações das restrições de integridade que podem tornar um banco de dados não íntegro, e sim outros mecanismos do sistema gerenciador do banco de dados (controle de concorrência e transações). A figura 8 apresenta o estado incorreto de um banco de dados, resultante de violações das restrições de integridade, ou seja, o não cumprimento das restrições definidas no projeto de banco de dados.

Estudante

RA	Nome	SobrNome	DataAniv
200768	Pedro	Santos	12/02/81
937653	Maria	Souza	10/10/80
937653	Joao	Smith	02/03/79

Curso

Codigo	Nome
01	Física
03	C. da Computação
04	Sist. De Informação

Exame

Estudante	Grade	Curso
200768	K	
937653	1	05
937653	C	03
200888	A	04

Figura 08: Banco de Dados com Informações Incorretas.

Ao analisarmos a figura acima, podemos observar violações que ocorreram em detrimento a não aplicação das restrições que normalmente são garantidas pelo SGBD:

- Nas duas últimas tuplas da relação *Estudante* aparecem dados de dois estudantes com o mesmo registro acadêmico. Situação inaceitável, pois o registro acadêmico deve ser único para cada estudante (restrição de chave primária ou única).

- Na primeira tupla da relação *Exame* existe um valor igual 'K' para o atributo *Grade*, esse valor não é válido, pois o domínio desse atributo está semanticamente definido para aceitar somente valores entre 'A' à 'F'. Ainda no atributo *Grade* o valor que aparece na segunda tupla é do tipo número, o qual deveria ser alfabético (restrição de domínio). Outra violação é informação vazia no atributo *Curso* (restrição de vazio), sendo que o estudante deve prestar exame pelo menos em um curso.

- Na quarta tupla da relação *Exame* aparece um valor para o atributo *Estudante*, mas este não está contido no atributo *RA* da relação *Estudante*

(integridade referencial). O mesmo ocorre na segunda tupla, o atributo *Curso* possui um valor que não está presente no *Código* da relação *Curso*.

3.1.2 Restrições de Integridade Estrutural Aplicada ao Modelo Relacional

Restrições de integridade estrutural são tipos de restrições definidas pela cardinalidade das relações (1:1; 1:n; m:n) no modelo ER – Entidade Relacionamento. Alguns modelos semânticos apresentam restrições estruturais de forma contundente, é o caso do modelo proposto por Abrial (1974), *data semantics* que foi a idéia mestra para o desenvolvimento de modelos utilizados até o momento. A restrição de integridade estrutural possui duas características: a) restrição binária (para relacionamento binário); b) restrição de participação (total, parcial). Essas restrições contêm propriedades inerentes ao modelo relacional. A figura 9[2] apresenta um exemplo simples, onde a cardinalidade do relacionamento é de (1:n) "um-para-muitos".

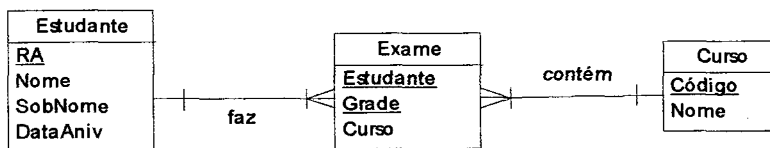

Figura 09: Modelo Entidade Relacionamento (visão lógica).

As restrições definidas para a propriedade especialização/generalização do MER, são restrições específicas e dependentes da semântica do modelo lógico utilizado, embora possa aplicá-la no modelo ER ou no modelo ER/E - Entidade Relacionamento Estendido. Para algumas especializações determinamos condições de um atributo (predicado definido) que pode especializar subclasse(s) de uma superclasse.

Outras restrições podem ser aplicadas à especialização. A primeira é a restrição de *disjointness*, que específica qual a subclasse(s) da especialização deve ser separada.

A segunda restrição de especialização é chamada de restrição completa, a qual pode ser total ou parcial. A especialização total específica que qualquer entidade da

2: Notação James Martin (Martin e Finkelstin) ou Engenharia de Informações, proposta no inicio dos anos 80.

superclasse deve ser membro de outra(s) subclasse(s) da especialização. Na especialização parcial considera-se que alguma entidade da superclasse não satisfaz o predicado, ou seja, não esteja especializada na subclasse.

Naturalmente, a restrição correta é determinada pelo mundo real de um domínio de aplicação, onde se aplica especialização. Todavia, a superclasse é identificada através da generalização, e esse processo é total porque a superclasse é derivada da(s) subclasse(s), desde de que contenha somente as entidades que estão nas subclasses.

O número de regras de inserções e deleções aplicadas à especializações/generalizações são especificadas primitivamente. Apresentamos algumas dessas regras [ELM 2000]:

- Excluir uma entidade da superclasse implica automaticamente na exclusão da extensão da entidade nas subclasses;
- Na inserção de uma entidade em uma superclasse, implica que entidade é mandatária, e deve ser inserida em todas as subclasses para qual a entidade satisfaça o predicado definido;
- Inserir uma entidade na superclasse de total especialização implica que a entidade é mandatária e deve ser inserida no mínimo em uma das subclasses da especialização.

Para que não ocorra criação de regras precipitadamente, deve-se fazer uma lista completa dessas regras, principalmente para as operações de inserção e deleção.

A figura 10, semanticamente indica que um estudante pode ser de graduação ou de pós-graduação, mas não ambos. Mas quando o domínio da aplicação exigir o contrário, então o modelo deverá ser alterado abrigando mais níveis de herança para representar a nova semântica.

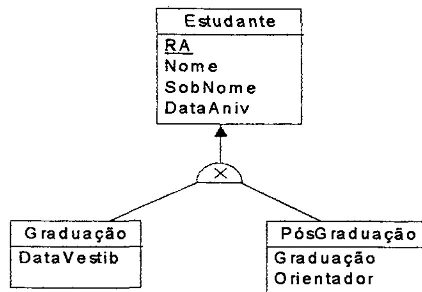

Figura 10: Exemplo de Especialização/Generalização.

3.2. Cumprimento das Restrições de Integridade Semânticas

A obrigação em manter o banco de dados íntegro e ao mesmo tempo atender o comportamento dinâmico das informações, toma necessário o uso de mecanismos para suplementar o controle semântico de integridade. Os principais mecanismos que tentam garantir a integridade semântica são: restrições pré-compiladas; asserções, coerção; e, *triggers*.

3.2.1 Restrições Pré-compiladas

As restrições pré-compiladas expressam as condições que devem ser satisfeitas para todas as tuplas das relações que sofrem atualizações do tipo: *INSERT, DELETE* ou *MODIFY*. O fato é, que se trata de uma espécie de averiguação, a qual verifica se a semântica definida entre a extensão da nova tupla (a ser inserida) ou da velha (a ser excluída) está sendo satisfeita. Esse tipo de verificação é anunciado pelo comando da SQL – *Structure Query Language* CHECK, em sua sintaxe.

```
Sintaxe: CHECK ON <nome da relação> WHEN <tipo de
            atualização> <qualificação sobre o nome
            da relação>)

Exemplos: CHECK ON Exame(Grade >= "A" AND Grade =< "F");
          CHECK ON HistSal(SalAtual < SalNovo);
```

Quando a verificação envolve mais de uma relação, então a restrição pré-compilada fica mais concisa e é denominada restrição pré-compilada geral, além de combinar funções agregadas.

```
Sintaxe: CHECK ON LIST OF <nome de variáveis>:<nomes das
         relações>,(<qualificação>)

Exemplo: CHECK ON e1.Estudante, e2.Estudante
         (e1.RA = e2.Ra IF (e1.Nome   = e2.nome AND
                          e1.SobNome = e2.SobNome))
```

3.2.2 Asserções

As asserções foram propostas formais sugeridas por Floretin (1974), com o objetivo de garantir a integridade semântica, bem como a consistência no banco de dados. Uma asserção é um predicado que expressa uma condição que desejamos que seja sempre satisfeita no banco de dados [SIL 1999].

```
Sintaxe: CREATE ASSERTION <nome-asserção> CHECK <predicado>

Exemplo: CREATE ASSERTION Restrição_curso CHECK
          (NOT EXIST * FROM Exame WHERE (SELECT Exame.Curso
                       FROM Exame,Curso
                       WHERE Exame.Curso = Curso.Codigo))
```

O principal problema para esse tipo de mecanismo é o custo de verificação das asserções, pois quando uma asserção é criada o sistema gerenciador de banco de dados certifica a sua validade. Se as asserções são satisfeitas, então qualquer modificação posterior no banco de dados será permitida somente quando a asserção não for violada. Em situações de complexidade ou de verificação de grandes volumes de dados pode surgir *overhead*, tornando inviável o uso desse tipo de controle de restrição de integridade. Devido a situações desse tipo, o uso das asserções não é recomendável, salvo em situações especiais, onde a verificação possa ser facilmente executada. Assim, tais situações têm levado muitos especialistas em desenvolvimento de banco de dados a evitar o uso de asserções.

3.2.3 Coerção

Esse mecanismo também consiste em rejeitar as atualizações (inserção, deleção e alteração) que possibilitam inconsistências no banco de dados. Essa opção combina a construção de algoritmos para garantir a integridade semântica do banco de dados, ou seja, há necessidade do uso adicional de ferramentas (linguagens de programação, algoritmos, API e SQL), além do SGBD para implementação desse tipo de restrição de integridade. O principal problema para esse mecanismo reside em encontrar algoritmos eficientes para esse propósito.

Existem dois métodos básicos utilizados no mecanismo de coerção: detecção de inconsistência (conhecido como pessimista ou pós-teste) e prevenção de inconsistência (conhecido como otimista ou pré-teste).

O método de detecção de inconsistência é aplicado para cada estado novo do banco de dados, isto é, após cada atualização um algoritmo verifica se a restrição desse novo estado foi violada. Se houver a violação o sistema gerenciador do banco de dados ainda tenta manter a integridade, modificando o estado inconsistente para outro novo estado através da compensação da ação, ou restaurando o estado anterior integro.

Por exemplo:

> *Atualização "1" é executada sobre "BD", causando a troca de estado do "BD0" para BD1. Em seguida o algoritmo de coerção verifica se o estado "BD1" é inconsistente. Se houver inconsistência o SGBD tenta manter a consistência pela compensação da ação gerando um outro estado "BD1'" ou restaurando "BD0" e desfazendo a atualização "1"* [OZS 1999].

Esse método pode ser ineficiente para um número considerável de execuções de transações, pois parte delas ficariam incompletas e conseqüentemente seriam desfeitas se a integridade falhar.

O método de prevenção de inconsistência denominado de *pré-teste* verifica através de um algoritmo de coerção todas as restrições de integridade relevantes à atualização, se a restrição em evidência não for violada, então a atualização no banco de dados é

efetuada. De qualquer forma nenhuma transação de atualização ficará incompleta em detrimento da violação da restrição de integridade, pois não há necessidade de desfazer as transações como no método de detecção de inconsistência. Por esse motivo o método pré-teste é mais eficiente que o pós-teste.

Exemplo:
```
UPDATE Exame "NewGrade"
  SET   Exame.Curso = Curso.codigo
  WHERE Exame.Grade >= "A"
  AND   Exame.Grade <= "F"
```

Ainda nesse método podemos utilizar um tipo especial de asserção, a asserção compilada, baseada na relação diferencial "R^+ U (R- R^-)". A asserção compilada é uma tríplice (R, T, C), na qual R é a relação, T é o tipo de atualização, e C é a asserção disposta sobre a relação diferencial envolvida no tipo de atualização T. Quando a restrição de integridade I é definida, então a asserção compilada pode ser produzida para a relação usada por I. Entretanto, a relação envolvida em I é atualizada por um programa u, a asserção compilada deve ser verificada para forçar I receber atualização somente do tipo u. Há duas vantagens nessa abordagem: Primeiro, o número de asserção para reforçar a restrição de integridade é minimizado, pois somente as asserções compiladas do tipo u, necessitam ser verificadas; O custo para forçar a asserção compilada é menor que forçar I, visto que as relações diferenciais em geral são menores que as relações base [OZS 1999].

3.2.4 *Trigger*

Trigger é um mecanismo baseado no paradigma *evento-condição-ação* (E-C-A), que pode ser utilizado na garantia da restrição de integridade semântica, embora possa ser aplicado em outras situações como em banco de dados ativo (apêndice A).

No paradigma E-C-A o *trigger* é ativado por um evento, ou seja, quando acontece um evento relevante e a condição for satisfeita, então a ação é executada. O exemplo a seguir é aplicado na restrição de integridade semântica, na qual o aluno somente poderá fazer exames no máximo em dois cursos diferentes.

Biblioteca Universitári
UFSC

0-344.593-7

```
Exemplo: CREATE TRIGGER ExameRefAluno
          AFTER INSERT ON Exame
           WHEN(NOT EXISTS
           (SELECT Estudante,COUNT(*)
              FROM Exame
              GROUP BY Estudante
              HAVING COUNT (*) > 2));
```

Considerações das Restrições de Integridade

Observamos que no cumprimento das restrições de integridade semânticas, quase sempre haverá necessidades de recursos adicionais além daqueles oferecidos pelo SGBD, e a implantação desses recursos estão fortemente ligados ao projeto do banco de bancos. Esses recursos podem ser linguagens de programação, *scripts,* API's, dentre outros.

A definição das restrições de integridade também obedece ao projeto do banco de dados, mas com distinção entre as restrições definidas transparentemente no próprio SGBD, em relação as restrições imposta por mecanismos externos. Dependendo do tipo de restrição definida, deve-se analisar o custo beneficio em relação ao seu cumprimento. Situações de extrema complexidade pode demandar custo de performance pelo fato do uso exaustivo dos recursos do sistema (SGBD).

No próximo capítulo veremos como é tratado o subsistema que garante o cumprimento das restrições de integridade em bancos de dados distribuídos. Atentaremos também pelo custo da verificação global das restrições distribuídas.

Capítulo 4

Restrições de Integridade em Bancos de Dados Distribuídos

As restrições de integridade (descritas no capítulo 3) contemplam um dos principais temas da tecnologia de banco de dados, que é o controle semântico dos dados. Os estudos das restrições de integridade em bancos de dados distribuídos baseiam-se em soluções aplicadas ao contexto centralizado. Porque há poucas soluções (SGBD's profissionais, ex. Oracle, DB2, Ingres, etc.) destinadas especificamente ao ambiente distribuído, sendo que aquelas aplicadas aos bancos de dados centralizados não oferecem uma performance elegante em detrimento às características dos bancos de dados distribuídos. O controle semântico envolve principalmente o projeto do banco de dados, algoritmos embutidos na linguagem de manipulação de dados e mecanismos externos desenvolvidos para apoiar a integridade dos dados distribuídos.

Gerenciamento da restrição de integridade em banco de dados distribuído é tratado como um subsistema do sistema gerenciador de banco de dados. A verificação da restrição de integridade envolve otimização em tempo de execução, quando isso ocorre nos banco de dados locais (execução da restrição local, idêntico ao modelo centralizado) não há dificuldade em garantir um certo nível de performance, mas quando a verificação é global (distribuída) envolve acesso remoto aos *sites*. Vários estágios de otimização devem ser incorporados ao projeto de gerenciamento da restrição de integridade distribuída.

Há diferentes tipos de propostas para garantir a restrição de integridade em banco de dados distribuído além daqueles disponíveis no SGBD, obviamente, por exemplo, *framework* baseado na engenharia de regras, teoremas, e algoritmos que apóiam a integridade. Mas a maioria das propostas culminam em mecanismos de cumprimento das restrições locais aninhados a um mecanismo de cumprimento global (as vezes o mecanismo global não aparece) [GUP 1993], [GUP 1994], [MAZ 1994], [CHA 1994], [ALO 1994], [CER 1995], [OZS 1999], que será um dos objetos de estudos deste

capítulo. Mas antes, abordaremos o uso de asserções aplicadas ao controle semântico do ambiente distribuído.

4.1. Asserções Distribuídas.

As restrições de integridade baseada em asserções são expressas em tuplas do cálculo relacional. Cada asserção é vista como uma consulta qualificada que envolve o armazenamento de dados em cada *site*, e esse método tende a minimizar o custo de verificação da restrição de integridade. Nessa estratégia aparecem três classes de asserções: asserções individuais; asserções orientadas a conjuntos; e asserções que envolvem agregações [OZS 1999]:

- As asserções individuais são do tipo simples, trata apenas de relação única e única variável. Elas referem a tuplas que são atualizadas independente do restante do banco de dados. Por exemplo, restrição de domínio (seção 3.1.1).
- Asserções orientadas a conjuntos incluem tanto restrições de relação única com diversas variáveis (dependência funcional), como também restrições de várias relações e diversas variáveis (restrições de chave estrangeira).
- Asserções que envolvem agregações são aquelas que exigem um processamento especial. Esse processamento é devido ao custo das avaliações dos agregados.

Escolher a classe de asserções ideal para característica distribuída envolve procedimentos e técnica, pois o uso de asserções demanda custo de processamento (seção 3.2.2). Lembrando que as relações são fragmentadas e podem ser localizadas em diferentes *sites*. Sendo assim, a definição da integridade através de asserções acontece com a operação distribuída, a qual é feita em dois passos. O primeiro passo é transformar uma asserção de alto nível em asserção compilada, (técnica discutida nas seções 3.2.1 e 3.2.2). O próximo passo é armazenar as asserções compiladas de acordo com a sua classe.

4.1.1 Asserções Individuais.

A função principal de uma asserção individual é certificar todos os *sites* que contém fragmentos das relações envolvidas na asserção. A asserção pode ser compatível com a relação de cada *site*, e essa compatibilidade é verificada em dois níveis: nível de predicado e nível de dado. A compatibilidade de predicado é verificada através da comparação do predicado da asserção com o predicado do fragmento. Veremos, uma asserção C não é compatível com o predicado do fragmento p se "C é verdadeiro" implica que "p é falso", e de outra maneira, é compatível com p. Se a não compatibilidade é encontrada em um dos *sites,* então a definição de asserção é globalmente rejeitada, porque tuplas do fragmento não satisfaz a restrição de integridade.

Em nível de dados, se a compatibilidade de predicado não foi encontrada, então a asserção é testada contra a instância do fragmento. Se não for satisfeita por aquela instância, então asserção também é globalmente rejeitada. Se a compatibilidade é encontrada, a asserção é armazenada em cada *site.* Note que o cumprimento de compatibilidade é restrito às asserções compiladas [OZS 1999]. Por exemplo, considerando a relação *Aluno(Codigo,* identificador do aluno*)* , fragmentada horizontalmente em três *sites* usando predicados (*p1,p2* e *p3).*

```
P1:  0 <= Codigo <  300
P2:  300 <= Codigo <= 600
P3:  Codigo > 600
```

Para o domínio da *assertion C:* "`Código < 400`". Asserção C *é* compatível com *p1* (se C é verdadeiro, *p1 é* verdadeiro) e *p2* (se C é verdadeiro, *p2* não é necessariamente falso), mas não é com *p3* (se C é verdadeiro, então *p3* é falso). Então, asserção C deverá ser rejeitada globalmente porque as tuplas do *site3* não satisfazem C, conseqüentemente a relação *Aluno* não satisfaz C.

4.1.2. Asserções Direcionadas.

As asserções direcionadas são multivaloradas e envolvem junção de predicados. Embora o predicado da asserção possa ser multirelação, uma asserção compilada é associada com uma simples relação. Então a definição da asserção é certificar todos os *sites* que armazenam fragmentos referenciados por essas variáveis. A averiguação de compatibilidade também envolve fragmentos da relação usada na junção de predicados. A compatibilidade de predicado não é usada aqui porque é impossível deduzir o predicado do fragmento *p*, igual a falso, se a asserções *C* (baseado na junção de predicado) é verdadeira. Então a verificação de compatibilidade de *C* pode ser contra os dados. Essa verificação de compatibilidade basicamente requer a junção de cada fragmento da relação, apontando que *R*, contém todos os fragmentos de outra relação, e *S*, está envolvido no predicado da asserção. Essa operação pode ser expansiva e, qualquer junção deverá ser otimizada pelo processador de consultas distribuídas. Há três casos que incrementam o custo de verificação:

- A fragmentação de *R* é derivada de *S* e baseada na semijunção de atributos usados na junção de predicados da asserção.
- *S* é fragmentado na junção do atributo.
- *S* não é fragmentado na junção do atributo.

No primeiro caso, a verificação de compatibilidade é barata desde que a tupla de *S* combine com a tupla de *R* e esteja no mesmo *site*. No segundo caso, cada tupla de *R* deve ser comparada com o principal fragmento de *S*, porque a junção do atributo equivale para a tupla de *R* que será usada para encontrar o *site* que corresponde ao fragmento de *S*. E no terceiro caso, cada tupla de *R* pode ser comparada com todos os fragmentos de *S*. Se a compatibilidade for encontrada para todas as tuplas de *R*, a asserção é armazenada em cada *site* [OZS 1999].

Apresentamos alguns exemplos para essa questão utilizando inserção na relação *Exame*. Dada a asserção compilada do tipo direcionada (*Exame, INSERT,C)*, onde C é:

$$\forall New \in Exame^{+}, \exists J \in (Aluno \vee Curso): NEW Matr = j.Aluno \vee NEW.Curso = j.Curso$$

Fragmentação da relação *Exame*: $(Exame \ltimes_{Aluno} Alunoi \ltimes_{Curso} Cursoi)$

A relação *Exame* é fragmentada usando predicado, onde *Alunoi* e *Cursoi* são fragmentos das relações *Aluno* e *Curso* respectivamente. E, neste caso cada tupla *NEW* de *Exame* aparecem no mesmo *site* das tuplas de *j*, como *NEW Matr* = *j.aluno* e *NEW.Curso* = *j.curso.* Desde de que a fragmentação de predicados seja idêntica a *C*, a verificação de compatibilidade não implica em comunicação.

Havendo fragmentação horizontal de *Aluno* e *Curso* baseado em dois predicados, onde:

```
P1: Matr < 3000 AND Curso < 3
P2: Matr ≥ 3000 AND Curso ≥ 3
```

Neste caso cada *NEW* de *Exame* é comparada com cada fragmento de *Aluno1* e *Curso1*, se *NEW.Matr* < 3000 e *NEW.Curso* < 3, ou fragmento de *Aluno2* e *Curso2*, se *NEW.Matr* ≥ 3000 e *NEW.Curso* ≥ 3.

Uma outra situação é a questão de dependência funcional de uma simples relação (*Curso*) (*Curso, INSERT, C)*, onde *C* é:

$$(\forall e \in Curso(\forall NEW1 \in Curso)(\forall NEW2 \in Curso)$$
$$(NEW1.Codigo = e.Codigo \Rightarrow NEW1.Curso.Nome = e.NomeCurso) \wedge$$
$$(NEW1.Codigo = NEW2.Codigo \Rightarrow NEW1.Curso.Nome = NEW.Curso.Nome)$$

Observando a verificação da asserção *C*, define-se a restrição entre as tuplas inseridas (*NEW1)* e as existentes (*e)*,(segunda linha) enquanto que na terceira linha, a verificação de restrição é realizada entre as próprias tuplas inseridas, através das duas variáveis (*NEW1* e *NEW2*) declaradas na primeira linha.

Considerando uma nova atualização para relação *Curso*. Primeiro a atualização da qualificação é executada por um processador de consultas que retorna uma ou duas relações temporárias, e nesse caso é uma asserção individual. Essas relações

temporárias são enviadas a todos os *sites* para serem armazenadas. Assumimos que a atualização é um comando de *INSERT*. Então cada *site* armazena o fragmento do *Curso* para reforçar a asserção *C* porque *e* é universalmente quantificado, *C* pode ser realizada pelos dados locais de cada *site*. Isto implica em dois fatos $\forall x \in \{a_1,.....a_n\}$ f(x) é equivalente $[f(a_1) \wedge .... \wedge f(a_n)]$ [OZS 1999]. Dessa forma, enquanto a atualização é submetida, cada *site* recebe a mensagem indicando que a asserção foi satisfeita e que esta é a condição para todos os *sites*. Se a asserção não é verdadeira para um dos *sites*, então esse *site* envia mensagens de erro, indicando que asserção foi violada. A atualização é invalidada, e a responsabilidade de decidir se o programa é rejeitado ou não é do subsistema da restrição de integridade.

Agora transformamos a asserção *C* (*Exame, INSERT, C)* descrita anteriormente em comandos de SQL.

```
SELECT NEW.*
FROM Exame+NEW, Aluno, Curso
WHERE COUNT(Aluno.Matr, Curso.Codigo
  WHERE(NEW.Matr = Aluno.Matr
  AND  NEW.Curso = Curso.Codigo))= 0
```

Note que *New** denota todos os atributos de *Exame*$^{+}$.

A estratégia desse caso é enviar novas tuplas aos *sites* para serem armazenadas nas relações *Aluno* e *Curso,* ordenando-as para a performance das junções, em seguida centralizar todos os resultados no *site query master*. Para cada fragmento de *Curso* e *Aluno* armazenados no *site,* haverá a junção deles com *Exame*$^{+}$, e os resultados dessa junção são enviados ao *site query master,* o qual efetua a união de todos os resultados. Se a união é vazia, então o banco de dados é consistente. Do contrário, a atualização promove um estado inconsistente do banco de dados. A rejeição do programa depende da estratégia utilizada pelo gerenciador de programas do sistema gerenciador do banco de dados [OZS 1999].

4.1.3 Asserções Envolvendo Agregação.

Além das características apontadas nas seções anteriores, as asserções que envolvem agregação consomem o maior custo de teste, porque requer o cálculo das funções agregadas. Geralmente as manipulações dessas funções, são *MIN*, *MAX*, *SUM* e *COUNT*. Outro detalhe é que, cada função agregada contém uma parte da projeção e outra da seleção. Para que essas asserções sejam utilizadas de uma forma mais eficiente é necessário que se utilize um mecanismo chamado de *concrete views* (proposto por Berstein e Blautein, 1982), isto é, a possibilidade de produzir asserções compiladas isolando a redundância de dados e armazenando as em cada *site*, e essa armazenagem deve ser associada a uma relação do *site*.

4.2. Verificação Global das Restrições de Integridade Distribuídas.

Como vimos na seção anterior, as asserções são mecanismos destinados a minimizar o custo da verificação das restrições de integridade distribuídas. Neste sentido, [GUP 1993] propôs um algoritmo que tende a otimizar a verificação global das restrições de integridade distribuídas, no qual estão associados: *two phase commit protocoll;* controle de concorrência distribuída; custo na comunicação da rede; e camadas de interfaces se os bancos de dados forem heterogêneos.

A essência do algoritmo é efetuar a verificação global das restrições através do acesso local aos dados, ou seja, o dado que será inserido no banco de dados local produz um teste condicional no local. Se esse dado satisfaz o teste, então baseada nessa prévia a restrição global será satisfeita. Essa idéia também foi sugerida por [ALO 1996] e [MAZ 1995].

4.2.1 Método para Verificação Local das Restrições de Integridade

Conforme [GUP 1993], a formalização da verificação e do teste condicional local, derivado da restrição global é baseada em uma tupla a ser inserida em uma relação local, onde teste local condicional é LTC($C, \bar{u}$) considerando uma restrição de integridade C:

$$(C)\colon \forall\bar{X}\,\forall\bar{Y}\,\exists\bar{Z} : [(L(\bar{X}) \wedge R_1(\bar{Y}_1) \wedge ... \vee R_k(\bar{Y}_k) \wedge g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})) \Rightarrow (S_1(\bar{y}'_1, \bar{Z}'_1) \wedge ... \vee S_n(\bar{y}'_n, \bar{Z}'_n))]$$

Sendo $\bar{u}$ a tupla a ser inserida na relação local L. O teste condicional local é o seguinte:

$$(\text{LTC}(C, \bar{u}))\colon \exists\,\bar{X}\,\forall\bar{Y}\,\forall\bar{Z} : [L(\bar{X}) \wedge (g(\bar{u}, \bar{Y}, \bar{Z}, \bar{c}) \Rightarrow g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c}))]$$

Se a relação L satisfaz LTC($C, \bar{u}$) então a inserção da tupla $\bar{u}$ em L não viola a restrição C. Note que somente a relação L é envolvida em LTC($C, \bar{u}$). O teste condicional não se refere qualquer relação remota $R_1,..R_k.S_1,...S_n$.

O teste é derivado em tempo de compilação por tratar $\bar{u}$ como um parâmetro em vez de usá-lo como o atual valor a ser inserido na tupla. Quando a tupla é realmente inserida na relação local L em tempo de execução, LTC($C, \bar{u}$) é instantâneo na inserção da tupla e na avaliação da mesma por usar a relação local. Na avaliação de LTC($C, \bar{u}$) quantificadores universais das variáveis $\bar{Y}$ e $\bar{Z}$ precisam ser eliminados da implicação ($g(\bar{u})$, $\bar{Y}$, $\bar{Z}$, $\bar{c}$) $\Rightarrow g(\bar{X}, \bar{Y}, \bar{Z}, \bar{c})$). A eliminação das variáveis em $\bar{Y}$ e $\bar{Z}$ resulta em um conjunto de restrições nas variáveis em $\bar{X}$ que serve como uma condição da seleção para cobrir a tupla na relação local [GUP 1993].

A definição acima é provada através do seguinte teorema:

> *Considere a restrição de integridade C e a tupla $\bar{u}$ inserida na relação L. Se o banco de dados satisfaz a restrição de integridade na asserção C antes da adição da tupla $\bar{u}$, e se o teste condicional local LTC($C, \bar{u}$) é cumprido pela relação local L, então o banco de dados satisfaz a restrição de integridade na asserção C depois da inserção da tupla $\bar{u}$.*

A sintaxe lógica da sentença (*C*) é a seguinte:

L representa a relação local.
$R_1,..R_k.S_1,...S_n$ representam as relações remotas.
$\overline{X} = \{X_1,...,..X_t\}$ é um conjunto dos quantificadores universais ($\forall$) ocorridos somente em L e g.
$\overline{Y} = \{Y_1,...,..Y_u\}$ é um conjunto de todas ($\forall$) as variáveis ocorridas somente em $R_1,..R_k.S_1,...S_n$ e g.
$\overline{Z} = \{Z_1,..., Z_v\}$ é um conjunto das variáveis ($\exists$) ocorridas somente em $S_1,...S_n$ e g.
$\overline{c} = \{c_1,...,...c_w\}$ é um conjunto de constantes ocorridas somente em g.
$g(\overline{X},\overline{Y},, \overline{c})$ é um conjunto dos iguais (=) e das diferenças ($<,>,\leq,\geq$).
$\overline{Y}_i \subseteq \overline{Y}$ é o conjunto das variáveis que ocorrem na relação $R_i, 1 \leq i \leq k$.
$\overline{Y}'_i \subseteq \overline{Y}$ é o conjunto das variáveis ($\forall$) que ocorrem na relação $S_i, 1 \leq i \leq n$.
$\overline{Z}'_i \subseteq \overline{Z}$ é o conjunto das variáveis ($\exists$) que ocorrem na relação $S_i, 1 \leq i \leq n$.

4.2.2 Gerenciamento das Restrições de Integridade em Projetos Distribuídos de Banco de Dados

Sendo que cada implementação de distribuição de dados possui características próprias, e as restrições de integridade obedecem a um domínio de aplicação específica, referenciamos uma arquitetura genérica proposta por [GUP 1996], o DCMS – *Distributed Constraint Management System*. Essa arquitetura facilita a detecção da inconsistência através da verificação das restrições de integridade. Os requisitos da inconsistência podem ser impostos a um simples *site*, ou envolver múltiplos *sites.* Isto é, as restrições de integridade podem referir-se a um simples banco de dados ou a múltiplos bancos de dados. Um outro propósito do DCMS é auxiliar no desenvolvimento de projetos distribuídos de banco de dados, sobretudo das restrições de integridade. O DCMS trata o gerenciamento local e global respectivamente conforme figura 11.

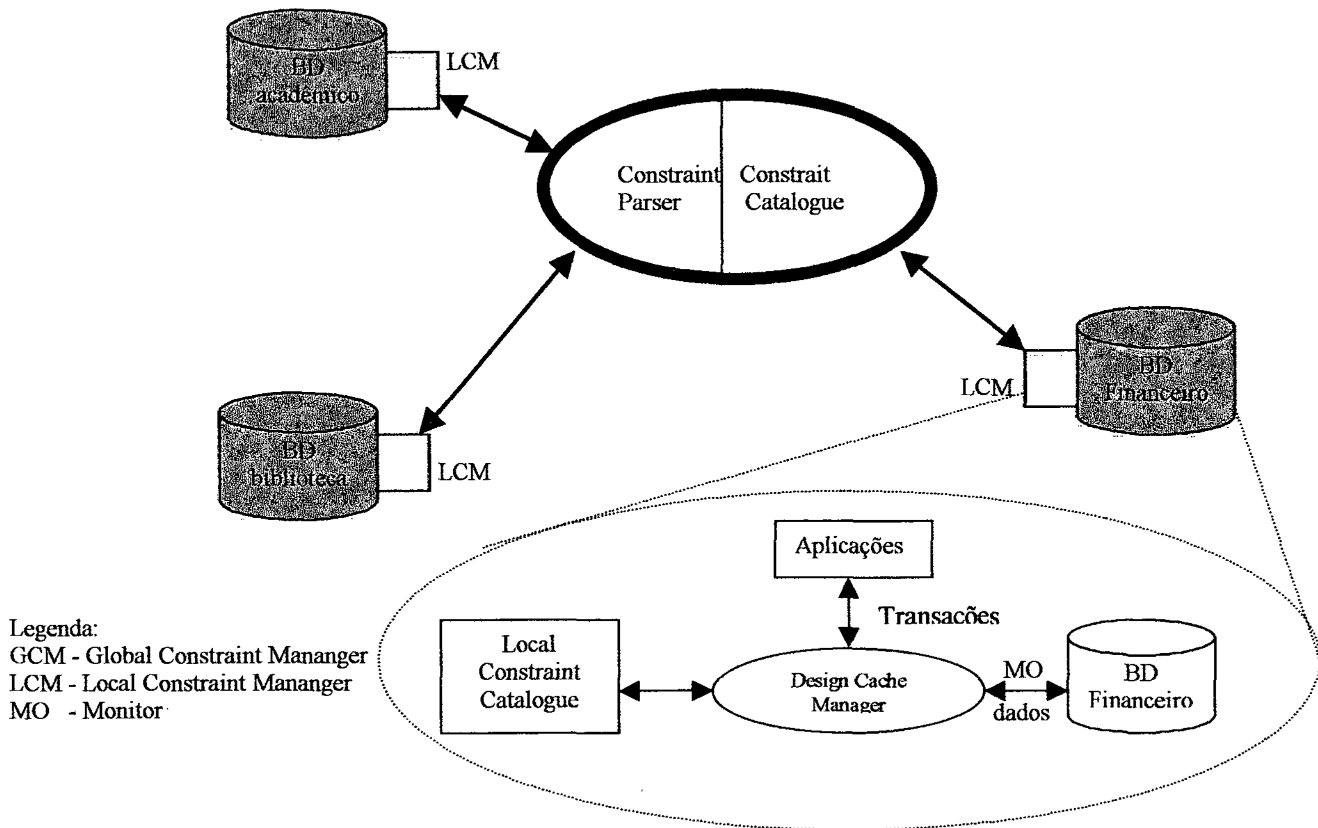

Figura 11: Arquitetura do *Distributed Constraint Management System* – DCMS.

A arquitetura proposta para o DCMS apresentada acima não é de um domínio específico de aplicação, e sim uma referência na elaboração de projetos distribuídos de bancos de dados, sobretudo no gerenciamento das restrições de integridade distribuídas. Assim sendo, veremos as funções de cada componente dessa arquitetura [GUP 1996]:

- O *design cache manager* em cada *site* suporta as interações das aplicações com o projeto do banco de dados. Ele permite a verificação de entrada e saída das transações e rastreia as modificações feitas no projeto. As restrições são especificadas, pré-processadas e armazenadas pelo *global constraint manager* (GCM).
- O GCM com fragmentos das restrições produz a otimização da informação, e distribui essas informações para os *sites* participantes. Individualmente nos *sites,* o *local constraint managers* (LCM) são responsáveis pela verificação

das restrições. A informação compilada enviada aos LCMs é armazenada em cada *site* nos *persistent catalogs*.

- Os *catalogs* são também usados pelo GCM para verificação global das restrições. Suportam a eficiente verificação das restrições, pois providencia acesso rápido a informação compilada no tempo de execução quando GCM e LCMs usam a informação para verificar as restrições. Os *catalogs* também suportam ainda consultas e atualizações das restrições.
- Os *monitors* trilham as modificações dos objetos no projeto do banco de dados, e detectando a ocorrência de operações que potencialmente violam as restrições de integridade. O LCM inicia o processo de verificação das restrições de integridade quando estimulado pelo *monitor.*

A verificação das restrições é mais expansiva na fase do gerenciamento, pois a qualquer momento será necessário a sua invocação quando um ou mais bancos de dados participantes da distribuição são atualizados. Os *monitors* dos bancos de dados informam ao *cache manager* das atualizações que potencialmente violam quaisquer restrições de integridade. O *cache manager* tenta evitar verificações redundantes e envia somente as modificações relevantes ao LCM. O LCM verifica cada restrição contra atualizações relevantes por executar um teste local de armazenado no *constraint catalog* em cada *site.* Se o teste local falhar, então o GCM é informado sobre essas atualizações. Uma *global query* (armazenada no *global constraint catalog*) é iniciada para verificar a restrição globalmente.

Conclusão

Percebe-se no decurso deste capítulo, que a preocupação principal é com o custo da verificação global das restrições de integridade sob o uso de asserções, entretanto há pesquisas que propõem algoritmos que tentam minimizar esse problema baseado na verificação local, como premissa verdadeira para verificação global das restrições de integridade distribuídas. O modelo genérico apresentado na figura 11 é um exemplo que pode ser adotado para essa questão. Esse modelo foi prototipado em C++ e API sob

SGBD Oracle, ou seja, houve adoção de recursos externos além daqueles do próprio SGBD.

Compreendemos que as linguagens declarativas possuem vantagens sobre as linguagens procedurais, na especificação das restrições de integridade quando existe um esquema global para a verificação das restrições de integridade. Por exemplo, o DCMS. Elas permitem um alto-nível de representação global das restrições e podem ter extensões.

No próximo capítulo apresentamos as características de replicação e distribuição de dados suportadas pelo SGBD Ingres II, sobretudo da definição e cumprimento das restrições de integridade distribuídas.

Capítulo 5

Restrições de Integridade Distribuídas no Ingres

O Ingres II é um SGBD distribuído oficialmente pela CAI™ – *Computer Associates Inc*[3]. absolutamente diferente em relação àquele que foi empacotado junto ao sistema operacional UNIX (sistema operacional também desenvolvido na *Bell Labs*). A primeira versão do Ingres - *Interactive Graphics and Retrieval System* surgiu em março de 1976 nos laboratórios da *Bell Telephone* o qual foi desenvolvido para DEC - *Digital Equipament Corporation* [STO 1976].

Neste capítulo verificamos as questões de definição e cumprimento das restrições de integridade distribuída suportadas pelo sistema gerenciador de banco de dados Ingres II (a partir de então será referenciado como Ingres). Mas antes voltamos a atenção às formas pelas quais o Ingres realiza a replicação e distribuição de dados e também os protocolos de comunicação suportados pelo SGBD Ingres.

5.1 Replicação de Dados no Ingres

O Ingres mantém cópias idênticas de dados em mais de um banco de dados, as quais podem estar no mesmo *site* ou em diferentes *sites* remotos distribuídos conectados por redes de computadores. Porém o Ingres atualiza somente uma base local e outra remota ao mesmo tempo. A facilidade de replicação no Ingres é movida por um de seus produtos destinado ao contexto distribuído, nesse caso o *Ingres/Replicator*. O *Replicator* necessita do Ingres *Server* para replicação distribuída, o qual inclui todos os componentes do cliente e do servidor, e nesse ambiente acontece a interação do SGBD local com o SGBD remoto. Além do Ingres *Server* a estrutura básica para replicação é composta de: *Ingres/Net*; e, *Ingres/Embedded*. *Ingres/Net* é responsável pelo acesso

---

3: www.cai.com

entre os nós remoto da rede, possui uma interface para configuração de protocolos de comunicação, e permite de forma transparente a transmissão dos dados entre o banco de dados de origem e o banco de dados de destino. *Ingres/Embedded* contém SQL embutido na linguagem *C* (ESQL/C) com ambiente de desenvolvimento para cada sistema operacional. O qual é utilizado para desenvolver aplicativos no *replicator server* que pode ser executado em qualquer outro servidor remoto compatível. *Replicator* também é compatível com ferramentas de desenvolvimento opcionais, como *OpenROAD* e *Ingres/Vision* [CAI 2000B].

Os níveis de replicação podem ser combinados em detrimento as necessidades de negócio da organização. O *Ingres/replicator* permite várias possibilidades de replicas, do tipo: um banco de dados inteiro; subconjunto de relações; subconjunto de colunas (particionamento vertical); e, subconjunto de linhas (particionamento horizontal). No mesmo sentido o *Ingres/replicator* pode ser configurado para atender as seguintes necessidades de replicação:

- De uma direção para um simples banco de dados de destino;
- De uma direção para múltiplos bancos de dados de destino;
- Ambas as direções entre dois bancos de dados;
- Muitas direções entre vários bancos de dados;
- Combinação de qualquer uma dessas opções.

A replicação pode acontecer continuamente, uma vez ao dia, a cada hora, ou sob demanda, essas opções são factíveis pelo mecanismo *assychrono* de distribuição de dados do Ingres. O sucesso de uma replicação está no planejamento envolvendo recursos de *hardware, software* e comunicação de dados (*Ingres/Net, ESQL/C, Ingres/Enterprise Access-Gateway* que se destina ao acesso de bases heterogêneas) indispensáveis para a realização da replicação dos dados. O *Ingres/replicator* possui uma interface amigável do tipo *Read-Write* com alguns sistemas gerenciadores de banco de dados (não Ingres), como: DB2, IMS, Rdb, DATACOM/DB e ALLBASE. Para os demais é *Read-Only*. Dessa forma o *Ingres/replicator* pode ser usado como *gateway*

para acessar bases de dados heterogêneas, principalmente em sistemas legados. A figura 12 representa uma simples arquitetura envolvendo conectividade da replicação.

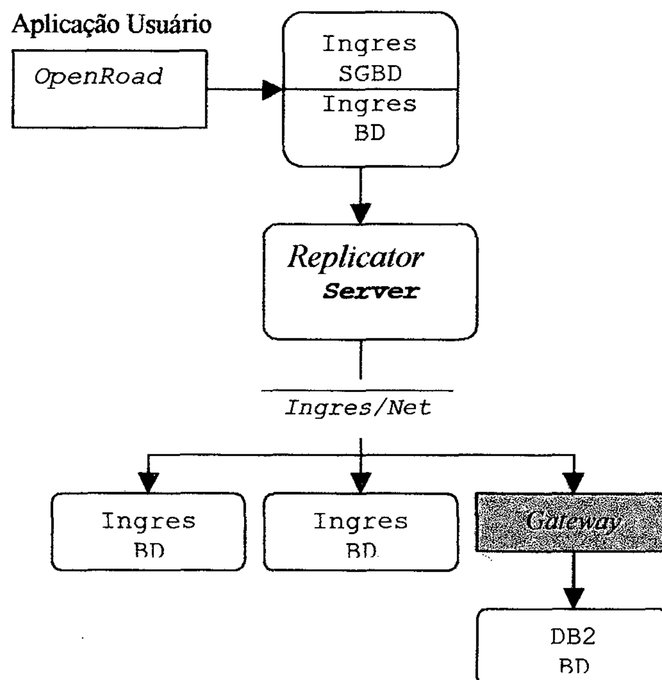

Figura 12: Arquitetura Simples de Replicação do Ingres.

O *Ingres/replicator* é o principal mecanismo da replicação de dados no Ingres, por tanto é composto por várias ferramentas, dentre as principais destacam: *change records; distribution threads; e, replicator server*.

Change Recorder, responsável em registrar todas as mudanças realizadas nas relações registradas para replicação, mantendo histórico de todas as transações da replicação, e uma fila (*input queue*) de transações que necessita ser distribuída.

Distribution Threads trabalha com SGBD para manter a informação da *input queue* e expande as especificações de quais dados necessitam ser replicados para o banco de dados de destino, a informação desses dados são armazenados na fila de distribuição. O *distribution threads* não realizada nenhuma distribuição, apenas prepara os dados para *replicator server.*

Replicator Server é um processo *stand-alone* que envia mudanças preparadas pelo *distribution threads* para os respectivos banco de dados de destino. Essas mudanças são realizadas usando *two-fase commit protocoll* com o objetivo de suplementar a integridade dos dados. Quando as transações são concluídas (*commited*) e as replicas estão seguras no banco de dados de destino, a replicação é removida da fila de distribuição. O r*eplicator rerver* também é responsável pelo tratamento do erro de colisão[4]. Além disso, o *replicator server* possui componentes adicionais usados no gerenciamento da replicação [CAI 2000b].

5.1.1 Possibilidades de Replicação no Ingres

Como o sucesso da replicação depende do planejamento da mesma, então o Ingres disponibiliza algumas situações de projeto para replicação de dados, como:

- *Central-to-Backup,* mantém uma réplica dos dados replicados com a finalidade de garantir a consistência da replicação quando ocorre uma falha no banco de dados de origem. Combinado com a replicação do tipo *peer-to-peer* (próximo item) permite ao *replicator* estabelecer a consistência recuperando os dados através dos bancos de dados remotos. Sem essa combinação o DBA – *Database Administrator* deverá restaurar a cópia de *backup read-only* manualmente.

- *Peer-to-Peer,* na replicação do tipo *peer-to-peer* cada *replicator server* joga uma mensagem, e recebe a função no ponto base contra outros bancos de dados de destino. Isso não é caracterizado como um relacionamento hierárquico entre vários servidores. Nesse modelo cada *replicator server* possui um funcionamento autônomo em seu próprio *site* usando suas próprias bases de dados replicadas. Todos os *sites* contêm as mesmas

4: A colisão ocorre na replicação de dados quando mais de um usuário atualiza a mesma tupla replicada em diferentes bancos de dados.

informações, pela razão de possuir sua própria cópia do banco de dados da organização.

- *Cascade,* A replicação em cascata ocorre quando o banco de dados de origem não tem a responsabilidade de distribuir dados a outros bancos de dados, então o banco de dados de destino que recebe a replicação move os dados para outro banco de dados de destino, assim a replicação é propagada até o destino final. É necessário que esteja projetado como *Full Peer*.

- *Central-to-Branch,* esse esquema ilustra subconjuntos de dados de um banco de dados central e designa para replicação nos *sites* ramificados com características *read-only*. Através de um centro de operações os dados são atualizados e enviados a esses *sites*. Esse esquema pode ser combinado com *peer-to-peer*.

- *Hub-and-Spoke,* nesse esquema o banco de dados central possui um relacionamento *peer-to-peer* com cada *spoke*. Implicitamente é uma replicação em cascata, na qual cada um dos *spokes* recebe a replica do dado, entretanto o banco de dados central ou qualquer outro banco de dados *spoke* sofre manipulações.

Nos projetos de replicações envolvendo o Ingres, há três conceitos que devem ser observados, independente do domínio da aplicação:

- CDDS – *Consistent Distributed Data Set,* é um subconjunto de dados consistentes (dados idênticos) mantidos em todos os banco de dados participantes da replicação. CDDS providencia métodos de definição e agrupamentos de dados assim que as entradas no banco de dados origem ou partes delas são replicadas em diferentes *sites*. Um dos principais objetivos do CDDS é definir quais dados foram replicados e onde eles estão.

- *Data Propagations Paths*, permite o *replicator* mover dados do banco de dados para outro conforme o caminho de propagação para CDDS. No caminho da propagação de dados, um banco de dados pode possuir um dos três papeis: *originator; local;* ou *target*. O *originator* é o banco de dados onde o usuário faz a mudança para uma relação replicada. *Local* é o banco de dados que propaga a mudança para o *target*. O *Target* é o banco de dados que recebe a mudança. Qualquer banco de dados no CDDS pode ser um *target* exceto o *originator* que possui seu próprio caminho, portanto *target* e *originator* não podem estar no mesmo caminho.

- *Target Types*, refere-se as funções que banco de dados realiza dentro do CDDS; o *target types* determina quais as mudanças no banco de dados devem ser replicadas e quais as colisões são detectadas. Os *target types* são definidos separadamente para cada CDDS, entretanto, um banco de dados pode ter mais de um tipo de *target* se ele conter mais de um CDDS. CDDS *target* suporta e faz cumprir os seguintes tipos de replicação: *full peer; protected read-only; e, unprotected read-only.*

5.1.2 Tratamento de Colisão no Ingres

A colisão no Ingres ocorre quando as trocas são destinadas ao mesmo registro em diferentes bancos de dados participantes do mesmo CDDS. Na condição de colisão, o *replicator* não consegue sincronizar os banco de dados sem extinguir os dados em colisão. O *replicator server* detecta a condição de colisão no momento da transmissão dos dados entre dois bancos de dados. Há três caminhos para detectar ou suspender a colisão no sistema de replicação: erros no *log file* do *replicator server* (`replicat.log`); relatório de colisão; e, mensagem do *replicator server (e-mail)*. *Replicator server,* por *default,* não trata as colisões e nem tenta resolvê-las. São efetuados o controle de detecção e as soluções das colisões através do CDDS *collision modes.* Quando a colisão ocorre, o *replicator server* identifica como um erro. Por exemplo, o servidor assume como erros: o registro não encontrado no banco de dados

de destino; ou a chave primária deveria estar no banco de dados de destino na transação de inserção. Se encontrarmos erros desse tipo no arquivo `replicat.log`, logo temos uma condição de colisão [CAI 2000b].

Há dois caminhos para tratarmos a colisão no Ingres: automática e manual. Ambas possuem vantagens e desvantagens. A colisão manual resolve as seguintes colisões:

- Determina a imagem correta do registro na colisão que deveria estar no sistema replicado;
- Encontra transação com chave duplicada para único registro;
- Utiliza o comando `set trace point DM32` contra a tabela que contém registros com erros;
- Atualiza o registro para estar com imagem correta;
- Atualiza o registro *shadow* para ter selecionado a transação chave;
- Remove qualquer comando de replicação que causa colisão aos registros provenientes da fila de distribuição;
- Repete os passos acima para cada banco de dados que participam da replicação e são afetados pela colisão.

Na solução de conflito automática, quando dois registros se deparam, um prevalece sobre o outro. Se a transação for de atualização ou de inserção um dos registros sobrevive à colisão por sobrepor o registro no banco de dados de destino. Se a transação for uma deleção o registro no banco de dados de destino é eliminado. Se o registro não sobrevive à colisão, as operações de replicação (*insert, update,* ou *delete)* para o banco de dados de destino são removidas da fila de distribuição. A solução automática de conflito é realizada apenas em nível de registro, ou seja, sobre escreve colunas que possuem valores incorretos, com valores corretos.

5.2 Distribuição de Dados no Ingres

O *Ingres/Star* é produto do Ingres que gerencia a distribuição de dados, o qual permite ao SGBD Ingres ser um sistema de banco de dados relacional distribuído. A distribuição contempla: acesso, armazenagem; e, processamento dos dados. Os requisitos básicos para modalidade implicam na combinação de sistemas operacionais, hardware, e outros produtos do Ingres como *Ingres/Net* e o *Ingres/Enterprise Access* (acessar dados heterogêneos).

Os componentes do *Ingres/Star* que habilita o Ingres ao contexto distribuído são:

- *Coordinator database* (CDB) contém um catálogo que o *star server* usa *para* manter o caminho dos objetos distribuídos. Quando um usuário requer uma informação, o *Ingres/Star server* acessa *coordinator database* que está associado a um banco de dados local, e através do sistema gerenciador de banco de dados local obtém a informação (embora o *coordinator database* seja um banco de dados Ingres, pode ser acessado como tal por outros bancos de dados, é importante que o acesso seja efetuado sempre via *Ingres/Star*).

- *iidbdd information* quando da instalação, define quais os bancos de dados serão distribuídos, qual deles será o *coordinator,* bem como a associação entre eles. *iidbdd information* contém informações de configuração que permite ao *Ingres/Star* ser mais eficiente ao realizar consultas distribuídas.

- *Links* são gerados opcionalmente para dados que ainda não foram registrados como objetos distribuídos. Através do comando `register as link` é adicionado a informação da localização do dado no catálogo do *coordinator database.*

Embora o banco de dados distribuído conecta-se a um banco de dados local, ele contém seu próprio conjunto de objetos. Esses objetos têm as mesmas características de

um banco de dados centralizado, como: relações; *views; procedures;* e, índices. Para distribuir relações aos bancos de dados que compõem um ambiente distribuído temos três opções: a) registrar relações existentes em banco de dados local utilizando comando `register as link`, o qual registra o nome e a localização das mesmas no CDB; b) criar novas relações através do comando `create table` diretamente no *star server,* as quais automaticamente são registradas no CDB e conseqüentemente compõem o banco de dados distribuído; e, c) criar novas relações em um banco de dados local e depois registrá-las no CDB. Ao registrar uma tabela, as suas propriedades (índices e domínios de atributos) também são registradas e preservadas [CAI 2000c].

A definição de dados para ambiente distribuído não difere muito de alguns comandos de DDL - *data definiton language* utilizados no contexto centralizado, os principais são: `create, drop, register e remove`. Na atribuição de nomes e definição de atributos para *star server* há algumas restrições, como; o nome das bases de dados deve ser único e de no máximo 32 caracteres; para linhas e atributos do tipo *varchar* o limite é de 4096 *bytes*; e, ao registrar uma tabela ou banco de dados, a definição de nomes deve ser em caracteres maiúsculo ou minúsculo, nunca a junção de ambos.

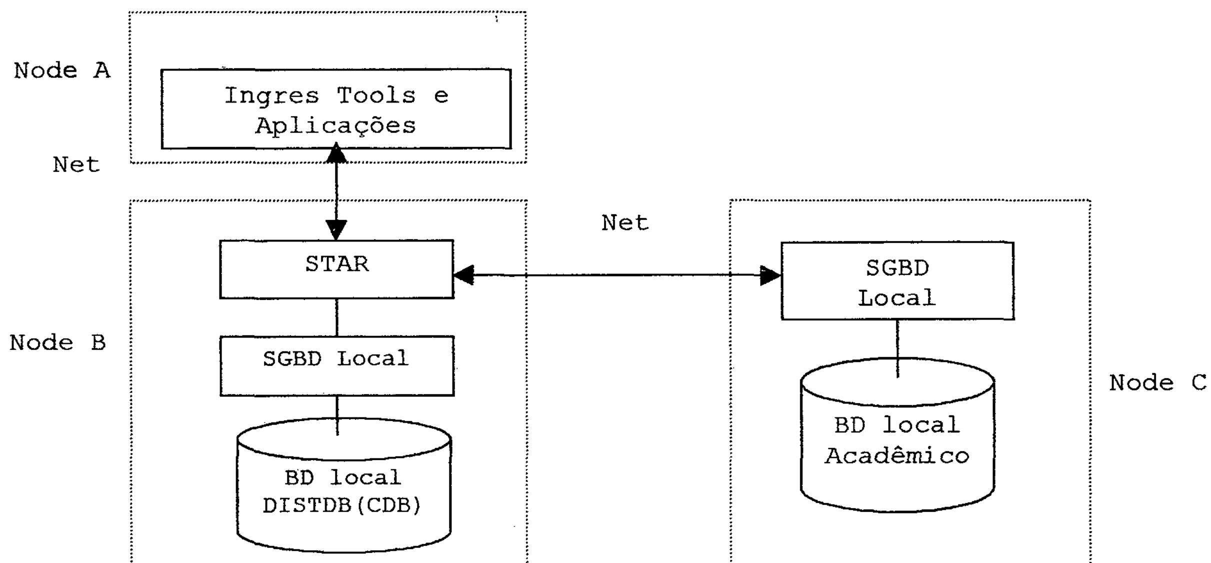

Figura 13: Arquitetura de Distribuição de Dados no Ingres.

No Ingres não há limites para a criação de banco de dados que comporá o sistema distribuído, a restrição é imposta pela plataforma utilizada. A figura 13 ilustra uma

arquitetura básica de distribuição de dados no Ingres. O Ingres disponibiliza uma espécie de *cache* compartilhado (*cache-sahring)* como opção, permitindo múltiplos servidores acessar um banco de dados através de um *buffer cache* comum. Dessa maneira as atualizações em cada servidor são realizadas imediatamente ficando disponíveis a outros servidores, desde que essas modificações estejam inscritas no *buffer de cache* compartilhado [CAI 2000d]. Essa opção aumenta a performance na distribuição de dados no Ingres, mas é recomendável que seja configurada em servidores multiprocessados.

Uma outra opção relacionada ao *cache* que visualiza uma performance ainda maior na distribuição de dados é o DMCM – *Distributed Multi-cache Management* que permite múltiplos servidores acessarem um banco de dados através de múltiplos *buffers* de *cache* privados. Ele habilita o servidor usar o *fast commit,* e com isso permite a execução de *commits* sem forçar a página de dados (*data pages)* do banco de dados. Essas atualizações também são visíveis a outros servidores [CAI 2000D].

O Ingres possui um conjunto de bibliotecas e extensões de programação (Ingres DTP - *Distributed Transaction Processing*) que permite aos usuários desenvolverem aplicações em conformidade com padrão *X/Open XA*[5] para serem executadas contra o Ingres *server.* O DTP suporta o modelo de processo *three-tie* (três camadas) que separa aplicação cliente, programação de aplicações servidoras e o gerenciamento independente de recursos.

5.3 Comunicação e Segurança no Modelo Distribuído do Ingres

Ao instalar os servidores para suportar um ambiente distribuído, o Ingres/*Ne*t deve ser configurado em cada servidor de banco de dados que compõem o ambiente. Os protocolos de comunicação suportados pelo Ingres/*Net* são: NetBios; TCP/IP –

5: O XA é um protocolo de interface bidirecional entre o gerente de transações distribuídas e o gerente de recursos. Ele é amplamente utilizado para ambientes distribuídos. Foi criado pelo consorcio *Open Group* denominado *X/Open*. *www.opengroup.org*

Transmussion Control Protocol/Internet Protocol ; SPX/IPX – *Sequenced Packed eXchange/Internet Packed eXchange* e DecNet.

Quanto ao acesso do banco de dados Ingres definido na criação do mesmo, ele pode assumir dois tipos: privado e publico. O acesso do tipo privado por *default* permite o acesso ao banco de dados somente ao DBA (ou proprietário), mas qualquer usuário pode solicitar seu direito de acesso. O acesso do tipo público por *default* habilita qualquer sessão a se conectar ao banco de dados. Entretanto o acesso desse tipo pode ser seletivo, onde os usuários podem ser classificados com níveis de acesso [CAI 2000a].

Ingres como a maioria dos bancos de dados prevê políticas de segurança para restringir o acesso dos usuários em determinados níveis como: acesso individual; em grupos; por papeis; e, públicos.

O Ingres garante a segurança dos dados pelo fato de possuir os requisitos do "*Orange Book*"[6], no qual é classificado atualmente como B1. Essa classificação é obtida porque o Ingres é portável ao *Sun* CMWs - *Compartmented Mode Workstations*. A segurança de vários níveis sobre as conexões Ingres/*Net* se da através do mecanismo CMW *MaxSix*.. Sem esse mecanismo a certificação de segurança pelo NCSC – *National Communications Security Evaluation* é C2, ou seja, (ITSEC F-C2) [WEB 2001a].

5.4 Definição das Restrições de Integridade Distribuídas no Ingres

O SGBD Ingres apresenta dois níveis de especificação das restrições de integridade, nas quais as restrições óbvias (integridade de chave, integridade referencial e integridade de domínio) são contempladas: restrições de integridade em nível de coluna; e, restrições de integridade em nível de relação.

Em nível de coluna, a especificação pode ser voltada a uma coluna, ou para um

6: Padrão de documentação do governo americano ITSEC – *Information Technology Security Evaluation Criteria*, o qual caracteriza a arquitetura de segurança na computação definindo níveis (A1 mais seguro ... D menos seguro).

conjunto de colunas. Ex. CREATE TABLE *Curso* (Nome CHAR (30) UNIQUE NOT NULL...), isso significa que a coluna nome da relação *Curso* assumirá valores não vazios e diferentes entre si. Em nível de relação, os dados pertencentes ao(s) grupo(s) de colunas que compõem a relação farão parte da restrição de integridade, mas há um limite de 32 colunas para cada tabela. O próximo exemplo demonstra que ao criar a relação *Aluno* restrições de vazios e de chave única são definidas para a relação.

```
CREATE TABLE Aluno
   (Nome        CHAR(30) NOT NULL,
   SobreNome    CHAR(20),
   Endereco     CHAR(50) NOT NULL,
   CONSTRAINT Uniq_Aluno UNIQUE (Nome,
                  SobreNome, Endereco));
```

Integridade de dados no Ingres é suportada por quatro situações de comandos: restrições, regras, eventos e comandos de integridade. As definições desse contexto são tratadas através da linguagem SQL padrão *ANSI 92 – American National Standards Institute,* as quais são aplicadas nas relações quando criadas (CREATE TABLE), ou após alteração (ALTER TABLE), alguns comandos são envolvidos nessa tarefa, como: *drop constraints, restrict, cascade, foreign key, primary key, references, check* [CAI 2000a].

As restrições são impostas no ato da criação da tabela ou através de modificações posteriores conforme a necessidade. O exemplo a seguir apresenta a sintaxe para estabelecer as restrições de integridade.

```
CREATE TABLE.....
   ([CONSTRAINT Constraint_name/ Table_constraint
   {CONSTRAINT Constaint_name] Table_constraint}...)
```

As regras no Ingres é uma característica pertencente a extensão do gerenciamento de conhecimento do Ingres, as quais são definidas por usuários que invoca uma *procedure* do banco de dados, sempre que a execução de um comando satisfaz a condição existente de uma regra, a regra é disparada e em seguida a *procedure* associada a essa regra é

executada. As regras são armazenadas junto das relações do banco de dados, conseqüentemente são aplicadas continuamente. O mecanismo de regras pode ser implementado para diversas propostas, uma delas para cumprir as restrições de integridade, quando utilizadas para essa finalidade quase sempre são associadas aos comandos *INSERT, UPDATE,* ou *DELETE* que aparecem nas aplicações ou são declarados pelos usuários.

```
CREATE RULE Insert_Aluno
    AFTER INSERT, UPDATE Aluno
    FOR EACH ROW EXECUTE PROCEDURE New_Aluno;
```

O *database events* do Ingres também faz parte da extensão do gerenciamento de conhecimento no Ingres, qual permite ao SGBD Ingres notificar outras aplicações que um evento específico ocorreu. Um evento pode ser qualquer tipo de programa detectável. No contexto da restrição de integridade podemos aplicar eventos combinando com regras para sistemas de tempo real, ou em situações que se faça necessário. O uso efetivamente de eventos no Ingres acontece em quatro etapas: criar evento; criar *procedure* (procedimentos armazenados junto das relações no banco de dados); criar regra e criar aplicação para monitorar o evento (exemplo a seguir).

```
CREATE DBEVENT [Schema.] Event_Name
 ...............
CREATE PROCEDURE
  BEGIN
   ............
  END
CREATE RULE Drill_Hot
   .........
  EXECUTE SQL REGISTER DBEVENT
   -----
 END.
```

O Ingres possui um outro mecanismo que pode ser definido para garantir a restrição de integridade, esse mecanismo é conhecido como comandos de integridade. Os comandos de integridade também são destinados a colunas e relações. Embora possamos utilizar o comando `CREATE INTEGRITY` associado aos comandos

CREATE TABLE e ALTER TABLE, onde a integridade é definida no momento da criação ou da alteração da relação.

```
CREATE INTEGRITY ON TableName [corrmame]
         IS Search_Condition

   CREATE INTEGRITY ON Exame IS
      Grade ≥ 'A' AND Grade ≤ 'F'

   CREATE INTEGRITY ON Curso IS
         Codigo ≥ 2
```

Não é aconselhável o uso de comandos de integridade (INTEGRITY) com CONSTRAINTS na mesma relação, pois comandos de integridade por *default* não permitem valores nulos, enquanto que CONSTRAINTS permite, isso pode comprometer a integridade do banco de dados.

5.5 Cumprimento das Restrições de Integridade Distribuídas no Ingres

A restrição de integridade nas bases replicadas possui os mesmos critérios de um banco de dados centralizado, ou seja, o banco de dados que sofrerá a replicação (destino) deve ter as mesmas definições para a restrição de integridade que o banco de dados que fornecerá a replicação (origem). Se o servidor de replicação encontrar comandos (CREATE TABLE E ALTER TABLE) diferentes entre a origem e destino ao replicar uma base, o *Ingres/replicator* envia como resultado um erro. O mesmo ocorre quando o servidor de replicação tenta replicar um dado para o banco de dados destino, e o encontra com a restrição de integridade violada [CAI 2000B].

O uso do *two-fase commit protocoll* reforça a integridade dos dados na replicação, pelo fato de que as duas fases do protocolo são usadas para concluir a transação quando duas bases de dados são atualizadas. O *Ingres/replicator* atualiza somente duas bases de cada vez, base local e a base remota. Durante a replicação o *two-fase commit* envolve os seguintes passos:

1- O servidor de replicação envia a replicação (tabela ou tupla) para banco de dados destino;
2- O servidor de replicação prepara o *commit* da transação entre a base local e a base destino;
3- Obtendo sucesso no passo 2 o servidor de replicação completa a transação realizando o *commit* em ambos bancos de dados, destino e origem;
4- O servidor de replicação repete o processo para o próximo banco de dados destino, para o qual a replicação precisa ser enviada.

Se ocorrer alguma falha durante o processo do *two-fase commit* (indisponibilidade da rede, ou do banco de dados) deve se identificar a causa, restabelecer os serviços de comunicação entre a base de origem com a base de destino e utilizar o arquivo de *log* da transação para concluir o *commit* da transação reiniciando o servidor (se for o caso) de replicação. O *Ingres/replicator* disponibiliza um utilitário para manipulação do arquivo de *log* da transação chamado *lartool*.

O uso de regras para o cumprimento da restrição de integridade possui uma vantagem proporcionada pelo fato de usar *procedures* (também conhecidos como subprogramas, que são procedimentos armazenados no banco de dados) através delas podemos desenvolver verificações de integridade mais complexas. No entanto, deve se ter cautela em utilizar essas regras, além do controle das *procedures* surge a clausula *AFTER* que determina o momento da execução da regra em função do comando. Essa clausula implica em um tipo de verificação da restrição (coerção) de integridade chamada de *pós-teste* ou pessimista (descrita na seção 3.2.3), a qual não é recomendável [CAI 2000a].

Os comandos de integridade `CREATE INTEGRITY` no cumprimento da restrição de integridade são eficientes porque estão diretamente ligados ao SGBD, mas se algum valor envolvido no comando compromete a restrição integridade declarada, automaticamente o SGBD aborta o comando e apresenta uma mensagem de erro.

A restrição de integridade distribuída no Ingres depende do funcionamento das transações distribuídas disponíveis no Ingres/*Star*. Desde então, não há nada de especial em relação restrição de integridade quando comparada a um banco de dados convencional. A transação distribuída envolve mais de um banco de dados local. Para o usuário do Ingres/*Star* uma transação distribuída é vista como uma transação de um simples banco de dados local, e os comandos de finalização da transação são os mesmos de um banco de dados local, `commit` e `rollback`. Mas uma transação distribuída pode envolver múltiplas transações locais e o Ingres/*Star* coordena essa situação como se fosse uma simples transação. Então todos os dados de uma transação distribuída são *commited* ou *roll back*, isso é possível através do conceito de atomicidade da transação, e o fato de que o protocolo utilizado pelo *Ingres/Star* é *two-phase commit protocol* (2PC) [CAI 2000b].

O Ingres/*Star* gerencia o protocolo 2PC da transação distribuída na seguinte forma:

1- Quando um usuário utiliza o comando `COMMIT`, *o* Ingres/*Star* notifica todas as bases de dados envolvidas na transação distribuída. Se todas as bases de dados indicam que estão preparadas para o *commit*, então o *Ingres/Star* decide pelo momento do *commit* da transação. As bases de dados locais ficam em estado de alerta para o *commit* esperando pela instrução do Ingres/*Star*.

2- Ingres/*Star* envia o *commit* para todos os *sites* envolvidos na transação e garante que todos eles realizarão o *commit*. Mas se a conexão é perdida com alguma das bases, entre o momento em que Ingres/*Star* decide pelo *commit* e o tempo em que a base de dados obedece a instrução *commit, o* Ingres/*Star* mantém a tentativa de completar a transação enquanto que a comunicação é restaurada e o *commit* é feito. *Ingres/Star* não retorna o controle ao usuário final enquanto todos os *nodes* tenham realizado o *commit*.

Se qualquer parte da primeira fase falhar, por exemplo, se *Ingres/Star* perde a conexão de rede com um *site* antes que todos os bancos de dados estejam preparados

para *commit*, então *Ingres/Star* executa `roll back` na transação para todos os *sites*, incluindo aqueles que já estavam preparados para o *commit*. Se qualquer parte da segunda fase falha, *Ingres/Star* eventualmente tentará realizar o *commit* da transação [CAI 2000b].

Não é em todas as situações que as transações distribuídas requer *two-phase commit:*transações que não efetuam atualizações em base de dados; transações que efetuam atualizações somente em uma base de dados; ou situações que não requer coordenação entre as bases de dados. Nesses casos o Ingres/*Star* consiste em enviar somente a mensagem de *commit* para cada banco de dados, ou seja, usando uma simples fase (primeira). Há também situações em que o acesso não permite o uso do *two-phase commit protocoll*, principalmente quando o acesso à base de dados é feito via Ingres/*Enterprise Access*, isso implica que algum banco de dados faz parte da distribuição, mas não permite atualizações, nesse caso para manter a sincronia das transações, o Ingres/*Star* simula *two-phase commit*. Mas, se mais de um *site* não suporta o protocolo e se envolve na transação de atualização, então Ingres/*Star* recusa a tentativa de atualização. Porque, primeiro ele envia a mensagem de preparação do *commit* aos bancos de dados que suportam o protocolo, e em seguida envia *commit* para o simples *site* que não suporta o protocolo, e finalmente envia *commits* para todos os outros *sites* que estão preparados para recebe-lo [CAI 2000b].

Conclusão

O SGBD Ingres possuí várias combinações e cenários para replicação de dados (seção 5.1). Também suporta transações distribuídas gerenciadas pelo produto *Ingres/star*.

Quanto a definição das regras de integridade, elas são definidas em conformidade com o projeto semântico do banco de dados independente do ambiente. Todavia o cumprimento dessas restrições de integridade é dependente das transações distribuídas alicerçadas pelo protocolo *two phase commit*. As duas fases do protocolo somente são

utilizadas quando há operações de inserção, deleção e atualização. O uso de regras ativas aumenta o poder de fogo no cumprimento das restrições de integridade distribuídas através das *procedures,* mas a verificação das restrições de integridade é do tipo pessimista (seção 3.2.3).

No capítulo seguinte demonstramos as características de replicação e distribuição de dados suportadas pelo SGBD Oracle8i , sobretudo da definição e cumprimento das restrições de integridade distribuídas.

Capítulo 6

Restrições de Integridade Distribuídas no Oracle

O Oracle8i é o quarto lançamento de produção do Oracle *server* lançado pela primeira vez em junho de 1997 (embora havia versões anteriores). Historicamente a Oracle® *Corporation*[7] foi a primeira a oferecer um SGBDR – Sistema Gerenciador de Banco de Dados Relacional em 1979 como projeto (Oracle Nasa) que se instalou na NASA – *National Aeronautics Space Agency* dos Estados Unidos da América. Atualmente é o SGBD mais utilizado pelas organizações. Ele é distribuído e comercializado pela *Oracle Corporation* e distribuidores autorizados [ABB 2000].

As restrições de integridade distribuídas no Oracle8i (deste ponto será denominado como Oracle) são tratadas neste capítulo sob o ponto de vista de definição e cumprimento. Mas antes se faz necessário a descrição das formas pelas quais o Oracle efetua a replicação e a distribuição de dados, bem como os protocolos de comunicação e segurança envolvidos.

6.1 Replicação no Oracle

Os mecanismos de replicação são embutidos no servidor Oracle, ou seja, não é necessário um novo artefato de hardware ou software para realizar a replicação. Quando houver necessidade de replicação, deve-se planejar o tipo ideal de replicação suportada no Oracle, configurar o servidor para que ele realize a função de replicação, e configurar o ambiente de rede através do produto Oracle *Net8*.

O Oracle aborda a replicação de dados com uma certa granularidade distinguindo a replicação em: replicação de objetos; replicação de grupos; e, replicação de *sites*.

7: www.oracle.com

Na replicação de objetos, qualquer atualização realizada em um objeto de um *site* é aplicada para todas as cópias em todos os outros *sites* participantes. Os tipos de objetos disponíveis para a replicação no Oracle, são: relações, índices, *views*, pacotes, *procedures*, funções, *triggers*.

A replicação de grupos trata da replicação de objetos usando a replicação de grupos, na qual constituí uma coleção de replicas de objetos, e os objetos na replicação de grupo são administrados juntos. O uso de replicação de grupos facilita o gerenciamento da replicação de objetos. Mas a replicação de grupos, e os esquemas do banco de dados necessariamente não se correspondem, entretanto a replicação de grupo pode conter objetos de múltiplos esquemas, e um simples esquema pode conter vários objetos em múltiplas replicas de grupos. Então uma replicação de objetos pode ser membro somente de um grupo de replicação.

Quando acorre a replicação de grupos em múltiplos *sites*, logo temos a replicação de *sites* em duas formas básicas: replicação *master site* (figura 14) e replicação *snapshot site*. No entanto, Oracle suporta os seguintes níveis de replicação: replicação *multimaster*, replicação *snapshot* e replicação híbrida (*multimaster e snapshot*).

6.1.1 Replicação *Multimaster*

A replicação *multimaster* é também chamada de ponto a ponto, permite acionar *sites* nos quais contêm servidores de banco de dados operando como *master* para gerenciar grupos de objetos replicados. As aplicações podem atualizar qualquer relação replicada em qualquer *site* na configuração *multimaster*, ou seja, convergir os dados de todas as replicas das relações para a garantir a consistência da transação global e integridade dos dados. Na replicação *multimaster* pode-se interromper o serviço de replicação para um *master* grupo a qualquer momento, essa opção facilita a manutenção das relações e uso de comandos DDL. Enquanto qualquer *master* grupo estiver parado nenhuma manipulação (DML – *Data Manipulation Language*) é permitida para os objetos do

master grupo. A figura 14 mostra a replicação de *sites* que operam como *masters* e gerenciam grupos replicados de objetos.

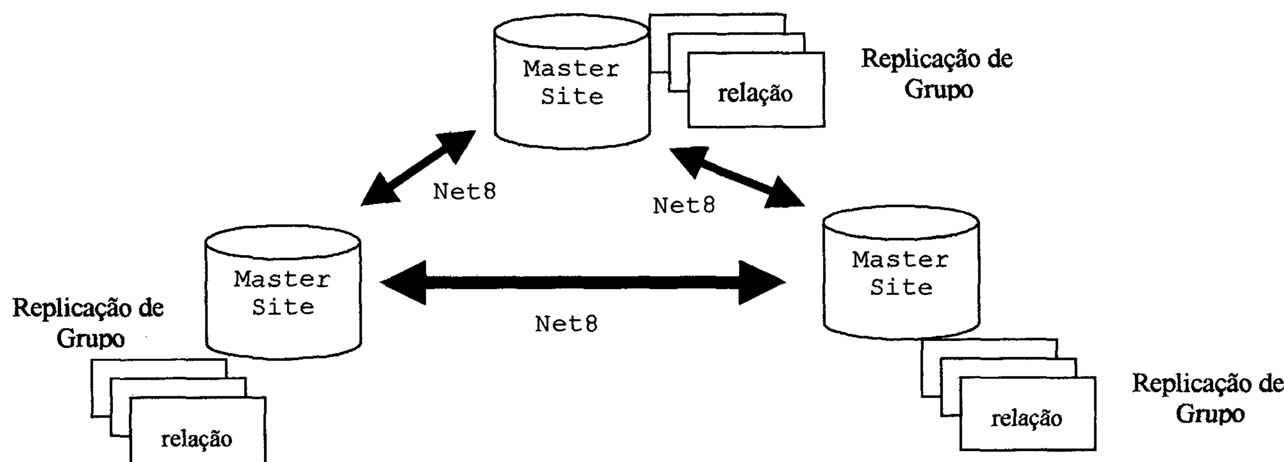

Figura 14: Replicação *Multimaster.*

Na replicação *multimaster* há três maneiras de implementarmos a cópia de dados: replicação *assynchronous* (mais utilizada); replicação *synchronous;* e, replicação procedural.

Na replicação *assynchronous* as atualizações realizadas em um *master site* se estendem algum tempo depois para os outros *sites masters* participantes da replicação. Ao contrário da replicação *assynchronous,* a replicação *synchronous* atualiza imediatamente as replicas em todos os *sites masters* participantes da replicação os quais estão contidos na transação. Se ocorrer alguma falha na comunicação a transação realiza o *rolled back* para todos os *sites* envolvidos na transação. Em situação semelhante, quando um dos *sites* não responde á comunicação nenhuma transação será completada enquanto a comunicação não for totalmente restabelecida [ORC 2000b].

A replicação procedural é utilizada para aplicações que possuem processamento em *batch* que realiza grandes trocas de dados em transações simples, normalmente usada para não sobrecarregar a rede. A replicação procedural replica somente a chamada para um procedimento armazenado que uma aplicação usa para atualizar uma relação. Ela não replica as modificações nos próprios dados. Há possibilidades de replicar pacotes

que modificam dados em todos os *sites*, após a replicação desses pacotes deve-se gerar um *wrapper* (uma espécie de capa) para os pacotes de cada *site*. Quando uma aplicação chama um pacote de procedimento no *site* local para modificar dados, o *wrapper* assegura a última chamada feita para o mesmo pacote em todos outros *sites* replicados [ORC 2000b].

6.1.2 Replicação *Snapshot*

A replicação do tipo *snapshot* pode realizar cópias completas ou parciais de relações *masters* instantaneamente. A replicação *snapshot* possui três propriedades: *read-only; updateable;* e, *refresh.*

Snapshot read-only é uma configuração básica que permite acesso *read-only* as relações originárias do *site master.* Aplicações locais podem realizar acesso aos dados na replica *snapshot* sem considerar a disponibilidade da rede. *Read-only snapshot* elimina a possibilidade de conflitos porque não realiza atualizações. A figura 15 ilustra a propriedade *read-only* da replicação *snapshot.*

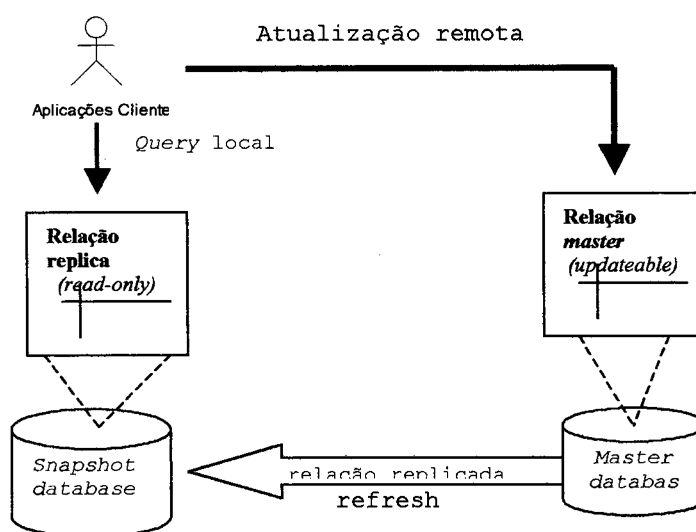

Figura 15: Replicação *Snapshot Read-Only.*

Updateable snapshot é uma configuração mais avançada que anterior cria um *snapshot* permitindo aos usuários realizar atualizações (inserção, atualização e deleção) nas linhas da relação master. *Updateable snapshot* pode conter somente um subconjunto de dados da relação master. Algumas propriedades do *updateable snapshot:*

- São sempre baseadas em uma simples relação.
- Podem ser renovadas incrementalmente.
- Propaga as mudanças realizadas em *updateable snapshot* para relação *master* remota. Se necessário, atualiza a relação *master* em cascata para os outros *sites master*.

A *Updateable snapshot* permite aos usuários consultar e atualizar um conjunto de dados local mesmo estando desconectado do *master site*. Ela requer menos recurso que a replicação *multimaster,* pois o *snapshot* pode estar em um banco de dados menor do tipo cliente (Oracle *lite*) com menos requisitos de espaço em disco e memória. [ORC 2000b].

O papel do *snapshot refresh* é garantir a consistência em relação ao banco de dados *máster,* para isso o Oracle providencia três métodos de *refresh:*

- *Fast refresh* usa *snapshot logs* para atualizar somente as linhas que tenham sido trocadas no último *refresh.*
- *Complete refresh* atualiza por inteiro as relações do *snapshot.*
- *Force refresh* quando não é possível realiza um rápido *refresh,* então o *force refresh* realiza um *complete refresh.*

A figura 16 caracteriza a replicação *snapshot updateable,* a qual permite atualizações nas bases de dados remotas e locais garantindo a consistência através do *snapshot refresh.*

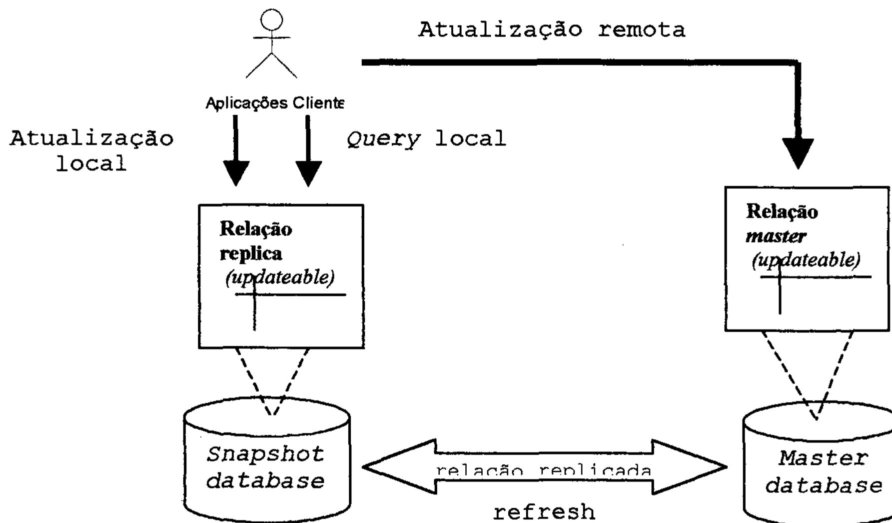

Figura 16: Replicação *Snapshot Updateable.*

6.1.3 Replicação Híbrida

A replicação híbrida é a combinação da replicação *multimaster* com a replicação *snapshot,* normalmente utilizada quando há diferentes requisitos na aplicação. A configuração *mixed* pode ter qualquer numero de *master sites* e múltiplos *snapshot sites* para cada *master*. Esse tipo de replicação une as propriedades contidas em ambas replicações *multimaster* e *snapshot* as quais foram descritas anteriormente.

6.1.4 Tratamento de Conflitos da Replicação Oracle

Ao projetar uma replicação de dados, deve-se considerar que em algum momento pode existir a concorrência na atualização de um mesmo dado nos *sites* participantes da replicação. Muitos projetos de sistemas de replicação tentam evitar conflitos em todos, ou em grandes porções de dados replicados. Entretanto, aplicações podem exigir que

alguns dados sejam atualizados em todos os *sites* a qualquer momento, e nesse caso, a possibilidade de conflito é eminente.

Oracle aborda os três tipos de conflitos de dados normalmente encontrados em projetos de replicação: *update conflicts; uniquenes conflicts;* e, *delete conflicts.*

- *Update conflicts* ocorre quando a replicação da atualização de uma tupla de uma relação com outra atualização para essa mesma tupla. Em outras palavras, o *update conflicts* acontece quando duas transações originárias de *sites* diferentes atualizam a mesma tupla aproximadamente no mesmo instante.
- *Uniqueness conflicts* ocorre quando a replicação de uma tupla tenta violar a integridade da relação, como a restrição de chave primaria ou chave única. Por exemplo, pode acontecer que duas transações de dois *sites* diferentes, e cada uma delas insere uma tupla nas respectivas relações replicadas com o mesmo valor para chave primaria. Nesse caso a replicação através dessas transações causam *uniqueness conflicts.*
- *Delete conflicts,* ocorre quando duas transações originárias de dois *sites* diferentes, uma transação exclui uma tupla, a outra transação atualiza ou também tenta excluir a mesma tupla. Neste caso a tupla não existe mais para ser atualizada ou excluída.

O Oracle possui maneiras automáticas de detectar e resolver conflitos na replicação. Cada *master site* no sistema de replicação, automaticamente identifica e resolve os conflitos na replicação quando eles aparecem. Por exemplo, quando um *master site* envia para fila uma transação *deferred* para o sistema de outro *master site,* procedimentos remotos são chamados no *site* receptor que automaticamente detecta a existência de conflito. O *master site* recepciona em seu sistema de replicação e consegue detectar os conflitos da seguinte forma [ORC 2000b]:

- O *site* receptor detecta *update conflict*, quando há qualquer diferença entre os valores antigos da tupla replicada (valores antes da modificação) e o valor corrente da mesma tupla no *site* receptor.
- O *site* receptor detecta *uniqueness conflicts*, se a restrição de unicidade foi violada durante a inserção ou deleção de uma tupla replicada.
- O *site* receptor detecta *delete conflicts*, quando não encontra a tupla para deletar ou atualizar porque a chave primaria da tupla não existe mais.

Além das opções automáticas de resolução de conflitos que o master *site* efetua, pode se ainda utilizar funções escritas em PL/SQL – *Procedural Language/Structure Query Language* as quais são implementadas de acordo com o problema existente. No entanto, a *Oracle Corporation* recomenda, que ao projetar uma replicação evitar qualquer possibilidade de conflito.

6.2 Distribuição de Dados no Oracle

A distribuição suportada no Oracle compreende três situações: sistemas de banco de dados distribuídos homogêneos; sistemas de bano de dados heterogêneos; e, arquitetura de banco de dados cliente/servidor (fora do escopo desse trabalho). Todas elas necessitam da configuração adequada do produto de rede Oracle *Net8* ao projeto de distribuição.

6.2.1 Bancos de Dados Distribuídos Homogêneos Oracle

Sistemas de bancos de dados distribuídos homogêneos são aquele que possuem dois ou mais bancos de dados Oracle, residentes em seus respectivos servidores e interligados por uma rede de comunicação. Para aplicação do cliente a localização e a plataforma do banco de dados são transparentes. A figura 17 evidencia uma aplicação de uma instituição de ensino distribuída suportada pelo ambiente Oracle.

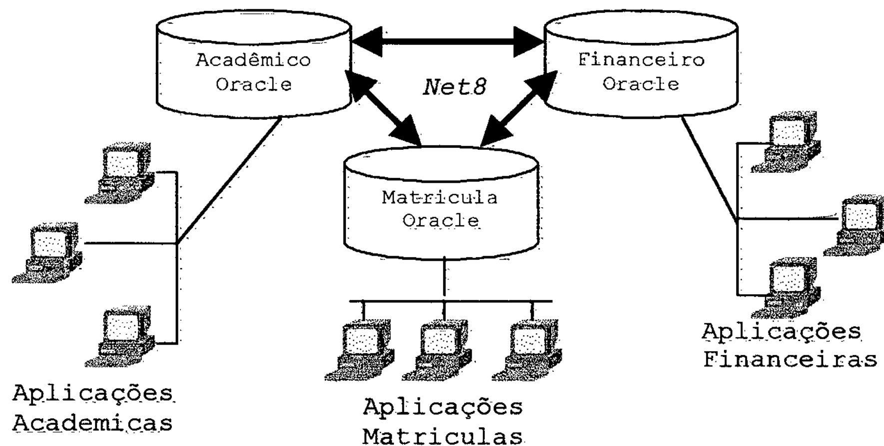

Figura 17: Exemplo de Sistema de Bancos de Dados Distribuídos Homogêneos.

6.2.2 Bancos de Dados Distribuídos Heterogêneos Oracle

Em um sistema de bancos de dados distribuídos heterogêneos, no mínimo um dos bancos de dados que participa da distribuição não é Oracle. Do ponto de vista da aplicação essa modalidade de distribuição é vista como um simples sistema de banco de dados local, ou seja, a distribuição e a heterogeneidade dos dados são transparentes. O Oracle disponibiliza serviços heterogêneos (SH) como um componente integrado ao servidor, o qual fornece uma arquitetura comum e mecanismos de administração de acesso a dados heterogêneos. A conectividade é estabelecida de duas maneiras:

a) Através de agentes transparentes, os quais são específicos aos sistemas que não são Oracle, e cada tipo requer um agente diferente. Esses agentes permitem a execução de comandos SQL.
b) Uso de conectividade genérica, através dos serviços de agente ODBC – *Open Database Conectivity* e OLE DB - *Object Linking And Embedding Database*, os quais compõe uma característica padrão do produto Oracle. A facilidade dessa opção é a de não haver a necessidade de uma configuração específica para o agente, simplesmente obter *drivers* de interface (ODBC ou OLE DB).

As características dos (SH), incluem: transações distribuídas; tradução da SQL; tradução do dicionário de dados; acesso as aplicações não Oracle através do dialeto SQL; acesso *stored procedure;* suporte a conjunto de caracteres *multi-byte; multi-threaded* agentes; entre outros [ORC 2000c].

O Oracle enfatiza a comunicação entre os *sites* através de seus *databases links*, um *database link* define o caminho de comunicação de um servidor de banco de dados Oracle para outro servidor de banco de dados também Oracle. O *link* é definido como uma entrada na tabela do dicionário de dados. Para acessar o *link* é necessário estar conectado ao banco de dados local que contém o dicionário de dados. O *link* é unidirecional, para acessar diversos banco de dados remotos é necessário criar *links* no banco de dados local para esses acessos. O banco de dados remoto não poderá utilizar o mesmo *link*, deverá definir seus próprios *links* e armazená-los em seu dicionário de dados. O *database link* pode ser compartilhado para reuso da conexão de várias sessões endereçadas ao mesmo banco de dados. A principal vantagem do uso de *database link* é permitir o acesso de um usuário local a um banco de dados remoto sem ser usuário do banco de dados remoto [ORC 2000c].

No ambiente distribuído Oracle, cada base de dados é identificada pelo nome global do banco de dados através de domínios da rede pré-fixados. O nome de cada banco de dados é definido por uma arvore começando pelas folhas até a raiz. Por exemplo, podemos ter domínios de redes, como: COM, ORG, EDU,... sob esses domínios é possível criarmos filiais, divisões, departamentos (vendas, produção, marketing ,....) e finalmente o próprio banco de dados. Então o nome para identificar cada banco de dados pode corresponder à hierarquia definida pela organização. Por exemplo, de acordo com figura 18, definimos a identificação das bases de dados para uma instituição educacional que possui múltiplos campi.

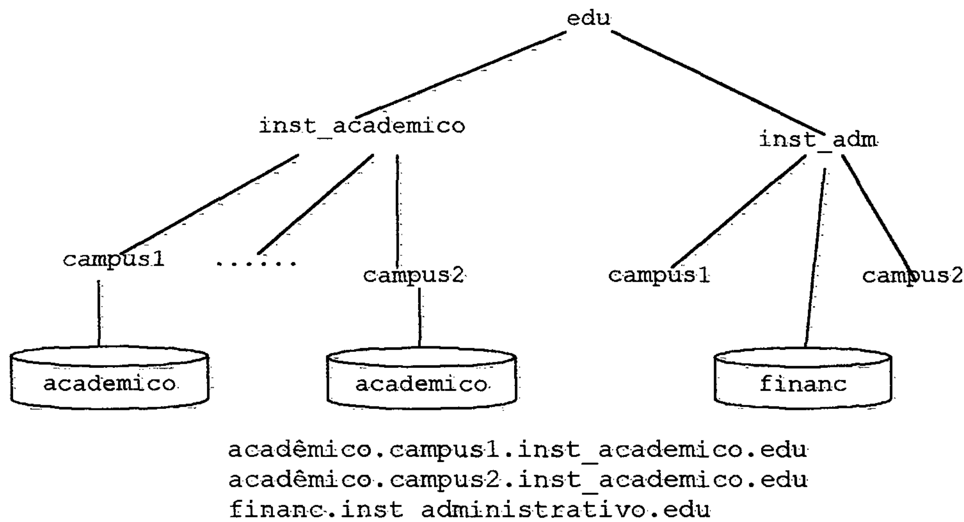

Figura 18: Domínio da Rede e Nomes Globais dos Bancos de Dados no Oracle.

A relação entre os nomes de identificação global dos bancos de dados é diretamente associada aos nomes dos *dabases links*, ou seja, tipicamente o *database link* possui o mesmo nome que banco de dados remoto no qual ele referência.

As permissões de acesso aos bancos de dados distribuídos no Oracle estão diretamente ligadas aos três diferentes tipos de *databases links* que podem ser criados como: privado, publico, e global.

Para obter um *database link* privado, cria-se o *link* no esquema específico do banco de dados local. Somente o proprietário do *database link* privado ou subprogramas de PL/SQL do mesmo esquema terá acesso a esse *link*, conseqüentemente acessará objetos do banco de dados correspondente.

Quanto ao *datbase link* publico, define-se um *database-wide link*. Todos os usuários e/ou subprogramas de PL/SQL do banco de dados que usa esse *link* acessa os objetos do banco de dados correspondente.

Quando a rede Oracle usa nomes próprios (figura 18) cria-se uma *network-wide link*. Nesse sistema o *database link* global é criado e gerenciado automaticamente para

qualquer banco de dados Oracle da rede. Usuários e subprogramas de PL/SQL de qualquer banco de dados que usa um global *link* pode acessar objetos do banco de dados correspondente [ORC 2000c].

A adoção do tipo de *link* depende dos requisitos das aplicações, então algumas considerações são relevantes:

- *Database link* privado, esse tipo de *link* é mais seguro que publico e o global, porque somente o proprietário (quem o criou) ou subprogramas contido no esquema, pode usar o *link* para acessar o banco de dados remoto.

- *Database links* público, quando vários usuários exigem o caminho de acesso ao banco de dados remoto Oracle, deve-se criar um simples *database link* público para todos os usuários do banco de dados.

- *Database links* global, quando uma rede Oracle usa Oracle *Names,* o administrador convenientemente gerencia globalmente os *databases links* para todos os bancos de dados do sistema, esse gerenciamento é centralizado e simples.

O Oracle permite o uso de *sinonyms* para esconder o *database link* do usuário (exemplo a seguir), possibilitando o acesso às relações de um banco de dados remoto como se fosse em um banco de dados local.

```
SELECT * FROM Aluno@Academico.campus1.inst_academico.edu;

CREATE SYNONYM Aluno
   FOR Aluno@Academico.campus1.inst_academico.edu;
```

6.2.3 Administração e Segurança dos Bancos de Dados Distribuídos Oracle.

O Oracle possui mecanismos de gerenciamento dos sistemas de banco de dados distribuídos como: *site autonomy*; banco de dados distribuído seguro; auditoria *database links;* e, ferramentas de administração.

Site autonomy permite que cada servidor participante da distribuição seja administrado independentemente dos demais. Ainda que vários bancos de dados trabalhem juntos, cada banco de dados é um repositório de dados em separado gerenciado individualmente.

Os bancos de dados distribuídos Oracle possuem a mesma segurança de um banco de dados centralizado incluindo: autenticação do usuário com senhas (da rede e do banco) e mais a autenticação externa com *kerberos* obtendo *end-to-end security*; conecções com *logins* encripitados entre cliente e servidor, e servidor a servidor; usuários globais podem ser autenticados através do protocolo SSL – *Service Security Layer;* serviços de autenticação da rede Oracle *Net8* com plataforma do servidor (Windows, Unix, Linux , Novell). A encripitação de dados disponível no *Net8* se da através dos algoritmos RC5 e o DES - *Data Encryption Standard.* Para o usuário de *database links* é determinado qual (is) banco(s) de dados ele pode acessar e nesse ambiente existem três categorias de usuários envolvidas no *database links* [ORC 2000c]:

- *Conect user* é um usuário local que acessa um *database link* no qual não foi especificado o *username* e *password*, mas deve ser especificado (nesse caso não precisa ser o usuário que criou o *link*, mas qualquer usuário que tem acesso a ele);
- *Current user* é um usuário do tipo global *database link* que pode ser autenticado pelo certificado X.509 e uma chave privada por estar junto ao banco de dados envolvido no *link*, *current user link* tem aspecto de segurança avançada do Oracle como opção;

- *Fixed user* são aqueles que o *username/password* compõem parte do *link* quando ele é definido, então o usuário precisa do *username* e *password* para se conectar ao banco de dados remoto.

Nos *databases links* permite-se efetuar auditorias nas operações locais e remotas realizadas pelos usuários que se conectam a eles, primeiramente na base local e em seguida nas bases remotas, cada operação é auditorado nos respectivos acessos.

O *Net8* é o ambiente que estabelece a conexão entre as bases de dados e também provê uma certa segurança na comunicação para a troca de dados. Através do *Net8* o Oracle suporta os seguintes protocolos: TCP/IP; TCP/IP com SSL; SPX/IPX; *Named Pipes*; LU6.2; e, *Bequeath*. Destaque para o protocolo TCP/IP com SSL, a comunicação com esse protocolo deve ser suportada pelos servidores envolvidos. O SSL armazena a autenticação de dados com certificados e chaves privadas no Oracle *Wallet*. Quando o cliente inicia a conexão com *Net8* para o servidor o SSL realiza uma verificação entre os dois (usando certificado), durante essa verificação ocorre a negociação entre o cliente e o servidor onde eles acompanham a cifragem, o conjunto de autenticação, e os tipos de integridade dos dados (da mensagem) serão aplicados nas mensagens cambiadas entre eles [ORC 2000d].

O Oracle possui ferramentas de administração integradas, como o *enterprise mananger tadabase administration tool*, ferramenta com interface gráfica que permite administrar um simples banco de dados centralizado, como também simultaneamente múltiplos bancos de dados.

6.3 Definição das Restrições de Integridade Oracle

As restrições de integridade estruturais no sistema gerenciador de banco de dados Oracle são definidas por comandos da SQL em conformidade com o padrão ANSI, principalmente quando do uso dos comandos `CREATE` e `ALTERNATE TABLE`. A seguir define se algumas restrições de integridade normalmente utilizadas em banco de

dados relacional (chave primária, chave única, integridade referencial e de domínio) que o Oracle utiliza independente da arquitetura (centralizada, distribuída).

```
CREATE TABLE Curso (
   Codigo NUMBER  (05),
   Nome   VARCHAR2(30));

ALTERNATE TABLE "Curso" MODIFY "Nome" NOT NULL;

CREATE TABLE Aluno(
  Matr   NUMBER  (05) CONSTRAINT Aluno_Pk PRIMARY KEY,
  Nome     VARCHAR2(30),
  SobNome VARCHAR2(30),
  Ender    VARCHAR2(50),
           CONSTRAINT Aluno_UK UNIQUE (Nome, Ender));
```

Restrição de integridade referencial no Oracle é definida por colunas comuns entre duas relações (ou um conjunto de colunas), onde a chave primaria ou única (*primary, unique key*) que compõem a relação *parent* e a chave estrangeira (*foreign key*) compõem a relação *child*.

A composição de chave primária, chave estrangeira ou chave única deve obedecer ao limite de 32 colunas. O Oracle permite que a chave estrangeira possua valores vazios, desde que não realize a verificação de compatibilidade com a chave a primária ou chave única, através das restrições de `CHECK CONSTRAINTS` e `NOT NULL`. A finalidade dessa opção é permitir a flexibilidade quando necessário em postergar a integridade referencial (recomenda-se ter critérios) [ORC 2000a]. Nesse caso deve ser definida ou alterada a restrições postergadas com os comandos `CREATE` e `ALTERNATE TABLE`. O exemplo a seguir define a restrições de integridade postergadas no Oracle.

```
CREATE TABLE Aluno (
  Matr NUMBER,
  Nome VARCHAR(40),
  CodCurso NUMBER REFERENCES (Curso),
CONSTRAINT Aluno_pk PRIMARY KEY (Matr) DEFERRABLE,
CONSTRAINT Aluno_fk FOREIGN KEY (CodCurso)
REFERENCES (Curso.CodCurso)DEFERRABLE);
```

O Oracle permite alterar o estado das restrições definidas, mas deve-se ter conhecimento das conseqüências. Também é possível revogar a restrição de integridade em nível de relação ou em nível de coluna através do comando `ALTER TABLE` com a clausula `DROP` (para coluna).

```
CASCADE
  ALTER TABLE Aluno
  DROP KEY,
  DROP CONSTRAINT Curso_Fk;

DROP TABLE Aluno CASCADE CONSTRAINTS;
```

Para as restrições mais complexas o Oracle pode definir *triggers*, o qual é suportado como um sistema de eventos. No contexto da restrição de integridade ele pode ser invocado quando da inserção, deleção ou atualização de relações ou *view* (relações temporárias). Um *trigger* no Oracle pode executar blocos de PL/SQL, procedimentos em *Java* ou na linguagem *C*. Porém há algumas recomendações quanto ao uso de *triggers:*

- Não definir *triggers* que duplique a funcionalidade embutida no Oracle, principalmente para o cumprimento da integridade de dados para as quais pode-se fazer do uso de declarativas.
- Não definir *triggers* com mais de 60 linhas ou 32 *Kbytes* de código, pois nesse caso é mais indicado o uso de *procedure.*
- Usar *triggers* somente centralizado, em operações globais eles são acionados por disparo de um comando, o qual é efetuado sem considerar quem o disparou (usuário ou aplicação do banco de dados).
- Não criar *triggers* recursivos.

```
CREATE OR REPLACE TRIGGER Aluno_Curso_Verif
   BEFORE INSERT OR UPDATE OF Curso ON Aluno
   FOR EACH ROW WHEN (NEW.Curso IS NOT NULL)
   DECLARE
    ......
BEGIN
    .....
END;
```

6.4 Cumprimento das Restrições de Integridade no Oracle

O Oracle utiliza o *constraint CHECK* (comando) para verificar as restrições de integridade. Esse comando tem por base expressões lógicas, sempre avaliando valores verdadeiros para expressão, quando ocorre o contrário o comando pode invocar o *roll back* da transação em questão.

```
CHECK (Colum_Name IS NOT NULL);

CHECK (Exame.Grade > = "A" AND
       Exame.Grade < "L");
```

O Oracle permite postergar (`CONSTRAINT DEFERRABLE`) a verificação da restrição de integridade principalmente quando a restrição não é satisfeita. Com isso é possível retardar a verificação da restrição enquanto finaliza a transação. Porém deve ser avaliado em quais características se deve utilizar esse mecanismo. Por exemplo: relações *snapshots*; relações que possuem grandes quantidades de dados que são manipuladas por outras aplicações, as quais podem ou não retornar dados na mesma ordem; e, atualização em cascata nas operações de chave estrangeira [ORC 2000a]. Mas mesmo assim a verificação da restrição de integridade pode ser adiada quando solicitada pelo usuário ou aplicação (SET `CONSTRAINT Aluno_fk DEFERRABLE`), ou seja, a verificação da integridade referencial entre `Aluno Curso` acontecerá somente quando a transação for finalizada.

Quando se definem restrições de integridade ao banco de dados, o objetivo é que esse banco de dados preserve a consistência de seus dados, assim como as regras de

negócios estabelecidas nesse banco de dados. Entretanto há determinadas situações em que o Oracle permite a suspensão temporária da restrição de integridade com finalidade única de *performance* nas seguintes tarefas:

- Quando se abastasse com grande quantidade de dados ou quando se utiliza nas relações o mecanismo SQL *LOADER*.
- Quando realiza operações em *batch*, na qual faz maciça troca de dados entre relações (troca entre relações temporárias e relações definitiva).
- Na importação e exportação de tuplas.

Há duas situações em que a suspensão pode ser aplicada:

a) na criação da relação com restrição suspensa;

```
CREATE TABLE Aluno
  (Matr NUMBER(6) PRIMARY KEY DISABLE);
```

b) quando da alteração da restrição existente na relação;

```
ALTER TABLE Aluno
   DISABLE PRIMARY KEY,
   DISABLE FOREING KEY;
```

Para desabilitar as restrições de integridade no Oracle, é imprescindível conhecer as estruturas de definição existentes, e nesse caso o uso do dicionário de dados é indispensável. De qualquer forma parece natural a suspensão temporária da restrição de integridade, mas ao considerar que enquanto existir violações das restrições de integridade o comando não é executado com sucesso, então a recomendação é que se exclua ou atualize as restrições de integridade. Lembramos que enquanto a restrição estiver suspensa o banco de dados pode ficar inconsistente, então essa opção deve ser utilizada com bastante cautela.

No cumprimento da restrição de integridade os *triggers* devem ser usados para garantir as restrições complexas de negócios, as quais não são expressas pelas declarações de restrição de integridade. Dessa maneira o uso de *triggers* no Oracle para esse contexto limita-se a:

- Ações referenciais de atualizações e declarações de nulos e valores *default*.
- Integridade referencial, quando a relação *parent* e a relação *child* estão em diferentes *nodes* em um banco de dados distribuído.
- Verificação de restrições complexas não suportadas em `CHECK CONSTRAINTS` [ORC 2000a].

A manutenção da integridade referencial no sistema distribuído do Oracle difere do centralizado. Porque não permite a definição de declarações de restrições de integridade referencial em *nodes* do sistema distribuído, ou seja, não há como estabelecer uma declaração de integridade referencial para uma tabela remota onde consta a chave primária ou única, bem como a referência dessa chave em uma relação local. No entanto pode se manter o relacionamento de relações *parent/child* em *nodes* distribuídos, o qual garante a integridade referencial através de *triggers* [ORC 2000a].

O processamento de transações possui um papel fundamental na restrição de integridade, principalmente para o contexto distribuído. As transações no Oracle são consideradas como unidades de trabalho constituídas por um ou mais comandos da SQL executada por um simples usuário. Para o modelo distribuído o processamento das transações é classificado da seguinte forma: transação remota que contém somente comandos que acessa um simples *site* remoto; e transação distribuída contendo comandos que acessam mais de um *site* remoto. Pode se realizar atualizações através de *procedures, triggers* definidos por subprogramas de PL/SQL.

O SGBD Oracle garante que todos os comandos de transação distribuída, ou não, realiza um dos dois comandos, `COMMIT` ou `ROLLBACK`. Os efeitos do andamento de uma transação deve ser invisível para todas as outras transações dos outros *sites*; essa transparência deve ser verdadeira para as transações que incluem qualquer tipo de operação (consultas, atualizações ou chamadas de *procedures* remotas). Oracle deve coordenar o controle das transações considerando as características da rede e manter a consistência dos dados, mesmo se rede ou sistema falhar [ORC 2000c].

Conclusão

Concluímos que o Oracle suporta vários tipos de replicação de dados (seção 6.1) e gerencia transações distribuídas através de protocolos padrões (seção 6.2), permitindo aos usuários a manipulação das bases de dados distribuídas. Também possui um bom nível de heterogeneidade.

O Oracle não utiliza produtos extras para implementar suas replicações e distribuições de dados, apenas necessita de configurações dos servidores e dos serviços de rede, bem como os protocolos envolvidos para a comunicação de dados.

As definições das restrições de integridade estruturais são efetuadas naturalmente através dos comandos de DLL (seção 6.3) independente do modelo (centralizado ou distribuído). Também permite a confecção de restrições mais complexas através do uso de *triggers,* procedimentos em PL/SQL e linguagens de programação C ou Java.

Quanto ao cumprimento das restrições de integridade distribuídas, limita-se basicamente aos mecanismos de gerenciamento das transações distribuídas do SGBD e aos recursos indispensáveis do protocolo *two-phase commit.*

No capítulo seguinte trataremos das mesmas características que foram abordas aqui, mas para SGBD DB2.

Capítulo 7

Restrições de Integridade Distribuídas no DB2

O DB2 *Universal DataBase* é um veterano e estabelecido sistema gerenciador de banco de dados, historicamente ele é datado em 1983, mas o fato que o DB2 é a raiz de um trabalho realizado nos laboratórios de pesquisas da IBM – *International Busines Machine* nos anos 70 [CHA 1998]. Atualmente o DB2 está entre os principais SGBD's profissionais utilizados pelas organizações e são comercializados pela IBM® *Corporation*[8] e representantes.

Neste capítulo são descritas as maneiras pelas quais o Sistema Gerenciador de Banco de Dados DB2 *Universal DataBase* v7 (doravante tratado como DB2) realiza a replicação e a distribuição de dados. Também investiga quais os protocolos envolvidos nessa tarefa, além de verificar como DB2 define e cumpri as restrições de integridade distribuídas.

A IBM disponibiliza produtos compatíveis com suas plataformas (OS/390, AS400, AIX – *Advanced Interactive eXecutive*, dentre outros), em função disso há situações que devem ser observadas ao abordar o contexto que será implementado a replicação assim como a distribuição de dados. O escopo investigado refere-se ao Sistema Gerenciador de Banco de Dados Relacional DB2 não considerando detalhes de implementação em relação à plataforma adotada.

7.1. Replicação de Dados no DB2

A replicação de bases relacionais é suportada é suportada pelo SGBD DB2 sob o uso do produto DB2 *DataPropagator*, que permite a replicação para qualquer banco de dados relacional DB2 (também pode ser usado em outros produtos da IBM, DB2 *DataJointer*, IMS *DataPropagator)* ou não, como *Microsof SQL Server e Sybase.*

8: www.ibm.com

O DB2 *DataPropagator* consiste em três principais componentes: *Administration Interfaces*, *Change-capture Mechanisms* e *the Apply program.*

O papel principal da *administration interfaces* é criar o controle das relações, as quais são armazenadas conforme o critério da replicação. Há duas interfaces destinadas ao *administration interfaces*, DB2 *control center* e o DJRA – *DataJoiner Replication Administration*. O DB2 *control center* é uma ferramenta de controle de banco de dados usada para administrar a replicação de dados entre dois servidores de banco de dados DB2. Ele mecaniza muitas funções como: criação de relações destino; controla essas relações quando especifica a informação destino; define relações e *view* como uma replica de origem; remove replicações de origem; duplica subscrição de relações para outros servidores; e, adiciona ou elimina chamadas de procedimentos ou comandos da SQL que são executados antes ou depois que o dado foi replicado.

O DJRA também é uma ferramenta de administração de banco de dados na replicação e administra varias tarefas, como:

- Cria controles das relações e os envia para suas origens, destinos e centro de controle;
- Define relções DB2, não DB2 e não IBM;
- Define *views* como replicações de origem
- Altera definições de relações DB2 n aorigem e no destino;
- Remove replicações origens e conjuntos de subscrição

O *change-capture mechanisms* realiza a captura de dados através de dois submecanismos: *capture program* e *capture triggers*. *Capture program* é utilizado quando a relação origem é DB2, então as mudanças ocorridas nas relações origem são capturadas através do *log* do banco de dados e armazenadas em relações temporárias. Já o *capture tiggers* criado automaticamente pelo DJRA captura as mudanças ocorridas nas relações origem do banco de dados não IBM (exceto *teradata, Microsoft access, e Microsoft jet)* armazenando-as em relações temporárias. *Capture triggers* são disparados quando um evento *(upadate, insert ou delete)* ocorre [IBM 2000a].

O *apply program* lê os dados diretamente das relações origem ou *views* para povoar relações de destino. Se as relações são de um banco de dados não IBM o *apply program* lê os dados através de um *nickname*. Se houver a necessidade de copiar as mudanças ocorridas nas relações então *apply program* lê as mudanças de dados que estão armazenadas nas relações temporárias e aplica essas mudanças nas relações destino. O *apally program* geralmente é executado no servidor de destino, mas pode ser executado em qualquer servidor da rede (origem, destino, e controle) que esteja conectado. Cada instancia de *apply program* pode ser executada no mesmo ou em diferentes servidores, cada *apply program* pode ser executado usando a mesma autorização (*user ID*). Cada *apply program* é associado a um servidor de controle, o qual contém o controle das relações. As relações por sua vez possuem as definições para as subscrições, que pode ser usado por mais de uma instancia do *apply program*. Por exemplo, se há um servidor de origem e dois servidores de destino, então separa se a execução do *apply program* para cada servidor de destino. As duas instancias do *apply program* compartilham o controle das relações, as quais deverão ter informações específicas e relatadas para cada instância aplicada [IBM 2000a].

7.1.1. Controles de Replicações do DB2

O DB2 possuí vários conceitos de controles usados na replicação, sendo os principais tratados a seguir.

Replications sources é basicamente é uma relação da qual se deseja realizar cópias de dados. Pode-se definir *replications source* para descrever informações que *change-capture mechanisms* usará, ou seja, essa definição concentra em determinar quais colunas serão replicadas e quais tipos de operações (*upadate, delete, ou insert)* serão tratadas.

Há duas características admitidas para coluna(s) da relação origem: *after-image* e *before-image*. O *after-image* contém o valor do dado na coluna da relação origem logo que o dado foi atualizado. No *before-image*, ao contrário do *after-image* o valor do dado

aparece na coluna da relação origem antes que o dado seja atualizado. Quando da definição em replicar pode-se optar pela captura do *after-image*, ou *before-image*, ou ambos, essa decisão depende do planejamento da replicação e do tipo de relação replicada. Por exemplo, uso do *before-image* é adequado quando a aplicação requer auditoria ou possui capacidade de *rollback*.

As cópias de dados realizadas pelo *apply program* são do tipo *full-refresh* e *differential-refresh*. O *apply program* copia dados da origem para o destino somente através do *full-refresh* ou do *differential-refresh*. No *full-refresh* o *apply program* realiza tarefas na seguinte ordem: a) elimina todas as linhas da relação destino; b) lê todas as linhas da relação origem; e, c) as copias para relação destino. Na cópia *differential-refresh* o *apply program* copia somente os dados modificados na relação origem para relação destino.

Antes de replicar os dados da origem, deve-se associar a origem com o destino, no qual serão replicadas as modificações ocorridas na origem. Essa associação é definida através do *subscripton sets* e *subscription-set menbers*, nos quais as informações das associações são armazenadas em várias relações de controle da replicação em questão. Uma *subscription set* deve ter uma *subscription-set menber* para cada relação destino ou para cada *view* também de destino. O *subscription set* assegura que todos *subscription-set menbers* são tratados da mesma forma durante a replicação: uma das duas modificações é aplicada para todos os destinos ou para nenhum deles. As modificações de dados para todos os *subscription-set menbers* em um *subscription set* são replicadas para relações destino especificadas em uma simples transação. A *subscription set* otimiza a performance porque as relações destino são processadas em conjuntos em uma transação contra o servidor destino. *Subscription set* também preserva a integridade referencial na replicação.

A replicação pode envolver somente um subconjunto da relação origem, usar uma simples *view* para reestruturar os dados da relação origem para relação destino, ou ainda usar junções e uniões complexas. Então pode se replicar algumas colunas ou linhas da relação origem em vez de replicar toda a relação para o destino, isso é

conhecido como particionamento de relações (vertical ou horizontal), no DB2 é chamado de *column subsetting* e *row subsetting.* A *column subsetting* é recomendada quando há colunas com grandes objetos (LOBs – *Larger Object Binary*), ou se o tipo de dado da coluna não é suportado pela relação destino. Quanto à *row subsetting,* ela é adotada de acordo com a necessidade de negócio da organização em relação a replicação.

Ao definir uma *subscription-set menber* deve-se especificar o tipo da relação destino que será usada na replicação. O DB2 apresenta disponibiliza os seguintes tipos de relações [IBM 2000a]:

- *User copy tables* são cópias de relações do tipo *read-only* da replicação origem, mas a replicação não controla a adição de colunas. Elas são consideradas como relações regulares, esse fato torna um bom ponto inicial para a replicação. Elas são mais comuns em relações do tipo destino.

- *Point-in-time tables* são cópias das relações do tipo *read-only* da replicação origem com uma coluna de *timestamp* adicionada. A coluna *timestamp* é originalmente nula. Quando as modificações são replicadas, valores são adicionados para indicar o momento que atualização foi efetuada.

- *Aggregate table* são relações também do tipo *read-only,* mas permite o uso de funções da SQL (como SUM e AVG) para sumarizar as entradas contidas nas relações de origem, ou as recentes modificações de dados ocorridas nas relações de origem. Há duas espécies de *aggregate tables: base aggregate table* e *change aggregate table. Base aggregate table* sumariza o conteúdo das relações origem, usa uma *base aggregate table* para trazer o estado da relação origem a um estado regular. A *base aggregate table* não traz informação da modificação. *Change aggregate table* trabalha com as modificações de dados no controle das relações e não com o conteúdo da relação origem. Usa-se uma *change aggregate table* para trazer as operações de modificações *(update, insert e delete)* realizadas no mesmo instante.

- *Consistent-change-data* (CCD) *tables* são relações que contém dados das transações *committed* (concluídas). Elas também possuem indicadores mostrando que as modificações ocorridas foram através de uma operação de *insert, delete* ou *update.* Mantêm juntos os valores antigos e novos dos dados. Cada tipo de CCD *table* (local ou remoto, completo ou não completo, condensado, ou não condensado, interno ou externo) possui uso diferenciado.

- *Replica* ou *row-replica table* são somente relações de destino que as aplicações às atualizam diretamente. Modificações feitas para *replicas* ou *row-replicas* são replicadas para relações de origem associadas, ou seja, a relação origem replica as modificações para outras *replicas.* As *replicas* são suportadas somente em banco de dados DB2. A *row-replica table* é um tipo especial de *replica* para o DB2 *DataPropagator* destinado ao *Microsoft Jet.*

- *User tables* não é especificada como uma relação destino, entretanto em *update-anywhere replication,* uma *user table* é automaticamente uma relação destino para *replicas* ou *row-replicas* que estão associadas a elas. O *user table* é um *parent replica,* e as cópias são *dependent replicas.* Um *parent replica* recebe atualizações de uma *dependent replica* e, se não for detectado conflito, ele replica as modificações para outros *dependent replica.* A *parent replica* é a origem primária de dados.

7.1.2. Métodos de Replicações Abordadas Pelo DB2

As replicações que podem ser estabelecidas para banco de dados DB2 obedecem a métodos de controles como: *synchronous replication; assynchronous replication; interval timing; event timing; e, on-demand timing* [IBM 2000a].

No método *Synchronous replication* as atualizações são entregues continuamente aos destinos. Quando há modificações nos dados de origem, esses são armazenados temporariamente e transferidos em seguida para o destino. As modificações são realizadas na origem somente depois que essas tenham sido replicadas no destino. Se por alguma razão as modificações não forem replicadas no destino, então não serão concluídas na origem.

O método *asynchronous replication* as atualizações são entregues em estágios. Quando a modificação é feita na origem, ela também é armazenada temporariamente e depois recambiada ao destino. O armazenamento temporário pode ser em minutos, horas ou ainda programado. Por exemplo, zero hora de cada dia (0:00), ou em uma data especifica de cada mês (dia 10).

Interval timing é um simples método que controla o momento da replicação. Há opções de data e hora para *apply program* iniciar a replicação de dados para o destino, e um conjunto de intervalos de tempo descrevendo a freqüência com que os dados serão replicados. O *interval timming* pode ser continuo formando um ciclo de replicação com alguns segundos de espera, mas o controle do parâmetro de espera é da aplicação. O intervalo de tempo usado no *apply program* depende do número de replicações que ele fará e os recursos disponíveis.

Event timing é o método mais preciso no controle de tempo da replicação. Para o uso do *event timing,* deve se especificar o nome do evento quando da definição do *subscription set* e definir o tempo em que ele será processado. Opcionalmente, pode se definir o tempo final do período; o *apply program* não executa nenhuma transação depois desse tempo, mas adiará as respectivas replicações até a uma nova data.

Na opção *On-demand timing* pode se replicar dados sob demanda usando o comando ASNSAT. Esse comando inicia o *apply program* e, se necessário, ele também executa o *capture program.* Cada programa termina depois de completar sua parte em um ciclo de replicação [IBM 2000a].

7.1.3. Cenários de Replicações Suportadas pelo DB2

*O DB2 contempla quatro tipos diferentes de replicação (*data distribution, data consolidation, update anywhere, e, occasionally connected*), porém há duas situações em que o DB2 não recomenda a replicação [IBM 2000a].:*

- Replicação de tempo-real, também chamada de replicação *synchronous,* ou seja, modificações ocorridas no sistema de replicação de origem são feitas imediatamente para relações de destino.

- Manutenção de servidor de *backup.* Não usa replicação *asynchronous* para manter acesso transparente servidor secundário quando o servidor primário não responde às requisições.

No *Data distribution* os dados contidos em um servidor de origem que sofrem alterações são replicados para uma ou mais relações residente em qualquer nó da rede distribuída. As relações de destino são *read-only*; entretanto não há necessidade a resolução de conflitos, porque não ocorrem conflitos durante a replicação. Esse tipo de configuração (conforme figura 19) é usado quando há necessidade de compartilhar dados entre vários *sites*, mantendo a performance das aplicações.

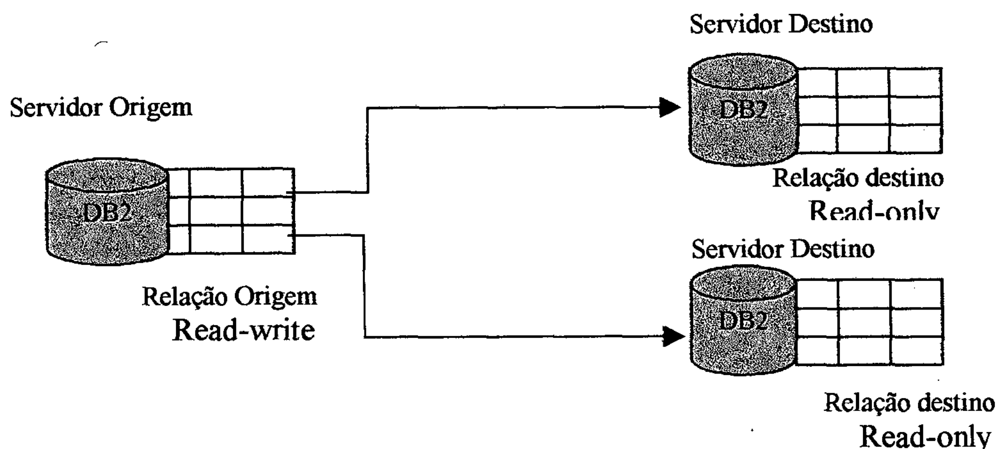

Figura 19: Replicação do Tipo *Data Distribution* do DB2.

Data consolidation é um servidor central utilizado como repositório central de dados para múltiplas origens. Essa configuração (figura 20) consiste em muitas relações

ou views de origem em uma relação destino com múltiplas views. As modificações ocorridas nos dados de origem são replicadas para um servidor de dados central com a propriedade read-only. Esse tipo de replicação comumente usado para sistemas de suporte a decisão, pois os dados podem ser analisados sem a concorrência dos servidores de origem produtores de dados. Nesse cenário, não há problemas de conflitos, pois cada origem atualiza um único conjunto de linhas na relação destino.

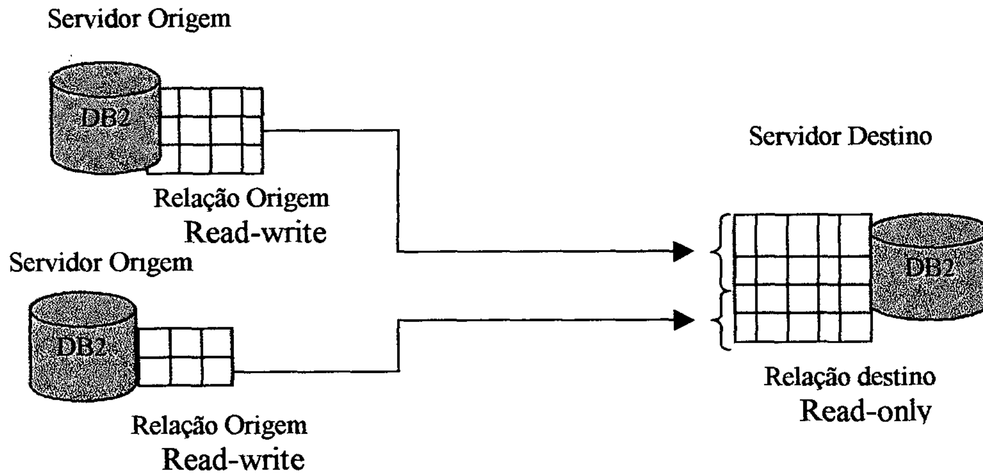

Figura 20: Replicação do Tipo *Data Consolidation* do DB2

A *Update anywher* é um tipo de configuração onde há duas situações: a) replicação com risco de conflito entre as relações destino (figura 21) e replicação sem o risco de conflito entre as relações destino (figura 22). Em ambas as situações produzem replicações *read/write,* ou seja, as modificações ocorridas nas relações destinos são aplicadas para as relações de origem, as quais mantêm os dados atualizados.

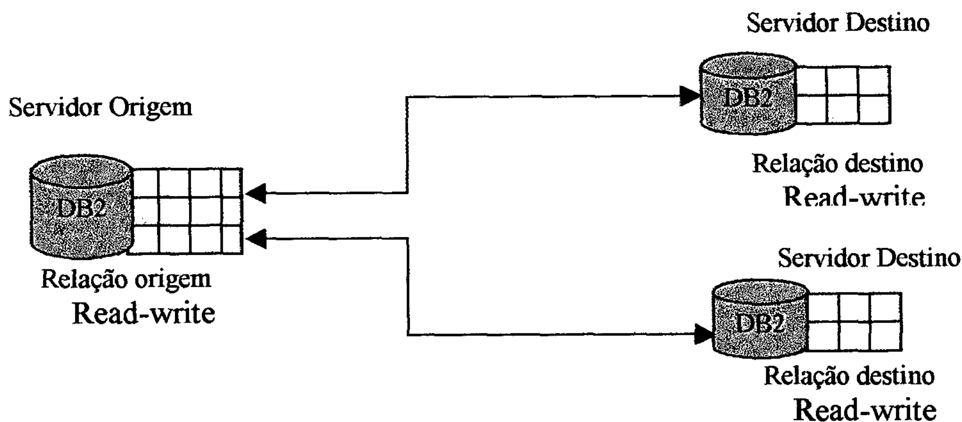

Figura 21: Replicação do Tipo *Update-anywhere* com Risco de Conflito.

A opção pelo risco de conflito ou não depende da necessidade do projeto de replicação. Quando há possibilidades de conflito, então deve se requerer a detecção de conflitos, porque todas as linhas de uma relação origem podem ser atualizadas em qualquer relação de destino. Pode-se ainda ignorar os conflitos e rejeitar qualquer atualização em conflito, porém há risco de perda de informações.

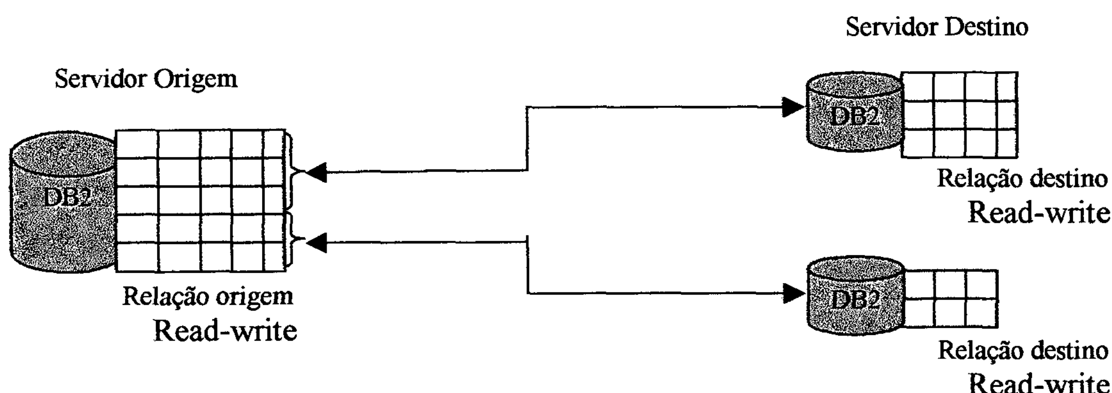

Figura 22: Replicação do Tipo *Update-anywhere* sem Risco de Conflito.

Occasionally connected é um tipo de configuração (figura 23) onde há flexibilidade de conexão e transferência de dados de uma origem primária sob demanda. Esse tipo de configuração permite ao usuário se conectar a origem primária de dados se houver uma sincronização mínima com um banco de dados local. A origem de dados não requer conexão contínua para administração da replicação. Esse tipo de replicação é destinado à base de dados móveis e *home offices*.

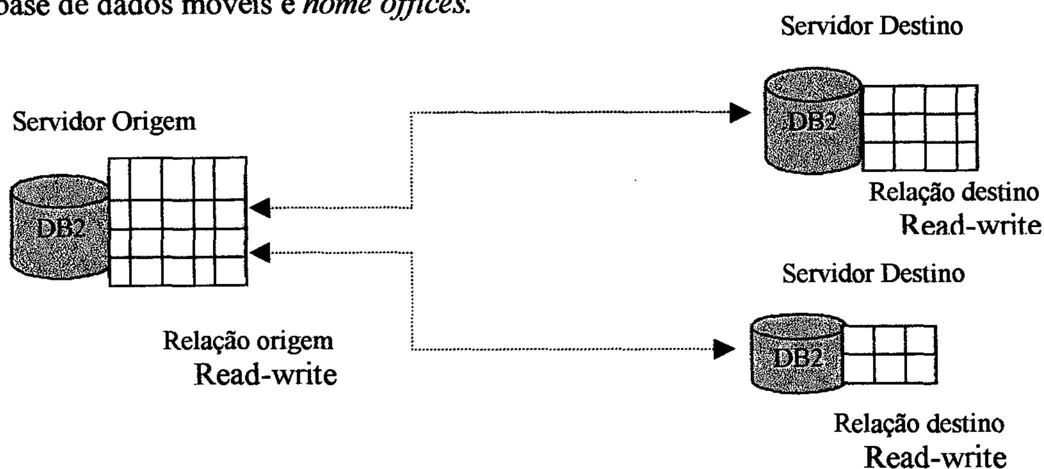

Figura 23: Replicação do Tipo *Occasionally Connected*

7.1.4 Resolução de Conflitos no DB2

A administração de conflitos pelo DB2 é bastante simples, ou seja, o conflito é apenas detectado e em seguida as transações envolvidas no conflito são desfeitas. As replicações que pode ocasionar conflitos, especialmente em combinações com a configuração *Update anywhere replication* devem ser evitadas. Embora DB2 *DataPropagator* detecta conflitos que ocorrem quando a mesma linha de uma relação é atualizada no sistema do *host* e no sistema do agente, e nesse caso nenhuma das modificações são replicadas. Se um agente faz atualizações que estão em conflito, estas serão descartadas durante a replicação para assegurar a integridade dos dados. A transação que estiver em conflito e todas as suas dependentes serão recuadas [IBM 2000a].

7.1.5. Restrições de Replicação no DB2

Há restrições específicas voltadas para sistemas operacionais, plataformas IBM e tipos de dados, das quais são referenciadas pelo DB2 *DataPropagator,* de ordem geral são [IBM 2000a]:

- Nome de relações e usuários pode conter até 30 caracteres, mas na replicação suporta apenas 18 caracteres;
- Dados compactados pelo EDITPROC ou FIELDPROC não são replicados;
- DB2 *DataPropagator* não captura chamadas para *stored procedure,* mas captura linhas atualizadas por *stored procedure;*
- DB2 *DataPropagator* suporta restrição de integridade referencial somente se as relações forem do tipo *user* ou do tipo *replica* (ambas definidas na *subscription-set);*
- DB2 *DataPropagator* não suporta as seguintes palavras chaves da SQL : CRETE TABLE para replica de relações que estão sujeitos à compensação: DELETE CASCATE, DELETE RESTRICT, e UPDATE RESTRICT.

- DB2 *DataPropagator* não captura atualizações realizadas por qualquer utilitário do banco de dados;
- DB2 *DataPropagator* não replica dados que estão encriptados;
- Tipos de dados definidos pelo usuário (diferentes dos tipos de dados suportados pelo DB2) deverão ser convertidos antes da replicação.

7.2. Distribuição de Dados no DB2

A distribuição no DB2 é regida por transações, as quais são tratadas pelo sistema gerenciador de banco DB2 como uma unidade de trabalho (*unit of work*). A *unit of work* pode assumir duas características: remota ou distribuída. Veremos a seguir as formas em que o SGBD DB2 gerencia e disponibiliza tipos de configurações para distribuição de dados.

7.2.1 *Remote Unit of Work*

Uma remota unidade de trabalho também conhecida como "usando apenas um banco de dados de destino na transação", (conforme figura 24) ela permite ao usuário ou um programa de aplicação ler e atualizar dados em uma localização através de uma *unid of work*. Ela suporta o acesso a um banco de dados dentro de uma *unid of work*. As características da *remote unit of work* são [IBM 2000b]:

- Suportar múltiplas solicitações (comandos SQL) através de uma *unit of work;*
- Suportar múltiplos cursores em uma *unit of work;*
- Cada *unit of work* pode suportar somente um banco de dados destino;
- O programa de aplicação da *unit of work* realiza *commits* ou *rolls back;*
- Dependendo da circunstância de erro o servidor de banco de dados ou DB2 *connect*[9] pode realizar o *roll back* da *unit of work.*

9: ferramenta que permite consultas em bases distribuídas e também o uso de transações que efetuam atualizações em bases distribuídas da família DB2

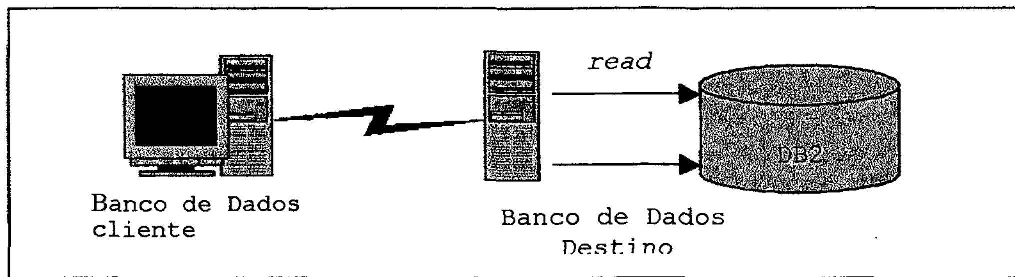

Figura 24: *Unit of Work* em um Simples Banco de Dados DB2.

7.2.1.1. *Distributed Request*

A *Distributed request* é uma função do banco de dados distribuído que permite aos usuários ou aplicações submeterem comandos SQL para referenciar dois ou mais banco de dados em uma simples transação. Por exemplo, junção entre tabelas de dois diferentes DB2 para subsistemas OS/390. Através do DB2 *connect* é possível realizar solicitações distribuídas entre SGBD's , como Oracle ou ainda realizar operações de união entre uma tabela DB2 e uma *view* Oracle[IBM 2000b].

Distributed request providencia localização transparente para os objetos do banco de dados. Se a informação contida (tabelas ou *view*) é movida, referencias para aquela informação (chamada *nicknames*) podem ser atualizadas sem qualquer modificação nas aplicações que a solicitaram. *Distributed request* também providência compensação para SGBD's que não suporta todos os dialetos do DB2 SQL. Operações que não são realizadas pacificamente sob o SGBD, e executadas sob DB2 *connect*.

Muitos fatores podem afetar a performance da *distributed request*. O mais crítico deles é assegurar a precisão da atualização da informação sobre os dados de origem, onde os objetos são armazenados em um catálogo do banco de dados global. Essas informações são usadas pelo DB2 *optimizer* (otimizador do DB2), e pode afetar as decisões nas operações de avaliação nos dados origem.

7.2.1.2. Atualizando Apenas um Banco de Dados

Quando há mais de um banco de dados distribuído envolvido em uma simples transação, a atualização ocorrerá somente em um deles, enquanto que os demais apenas sofrem leituras, então encontramos a seguinte situação conforme a figura 25.

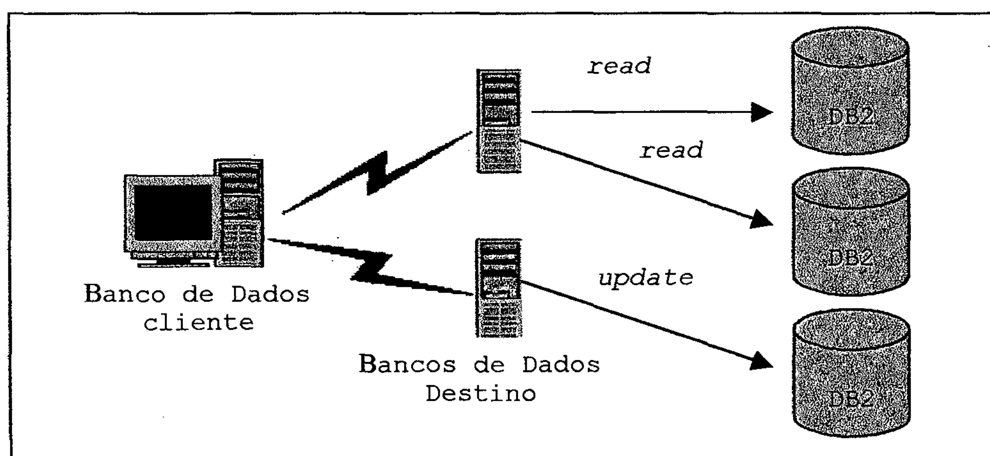

Figura 25: Usando Múltiplos Bancos de Dados em uma Simples Transação.

Quando há envolvimento de vários bancos de dados em uma simples transação, a instalação e administração do desenvolvimento são diferentes, e implica no número de banco de dados que serão atualizados na transação [IBM 2000c].

7.2.2. *Distributed Unit of Work* - DUOW

A *distributed unit of work* também conhecida como *Multisite update* ou *two-phase commit*, ela assegura que as aplicações podem atualizar dados em muitos servidores de banco de dados remotos com a integridade desses dados garantida [IBM 2000b]. Este suporte à DUOW é disponível para aplicações que utilizam SQL regular, bem como aquelas que usam o TM *Transaction Monitor* (produto que implementa várias interfaces como X/Open XA, MTS – *Microsoft Transaction Server*, e, vários outros).

Juntos a SQL nativa e os programas que utilizam TP monitor *multisite update* devem ser pré-compilados com *connect* utilizando a opção CONNECT 2 SYNCPOINT TWOPHASE (para usar o protocolo *two-phase commit* em duas fases). Ambos podem usar o comando SQL CONNECT para indicar qual (is) banco (s) de dados eles desejam usar para os demais comandos da SQL que acompanham as aplicações. Se não há TP monitor para o DB2, então o SGBD DB2 deverá ser usado para coordenar a transação [IBM 2000b].

Quando há dados distribuídos em múltiplos bancos de dados, e a aplicação exige que haja execuções de várias leituras e várias atualizações na mesma transação, então encontramos a seguinte situação (figura 26).

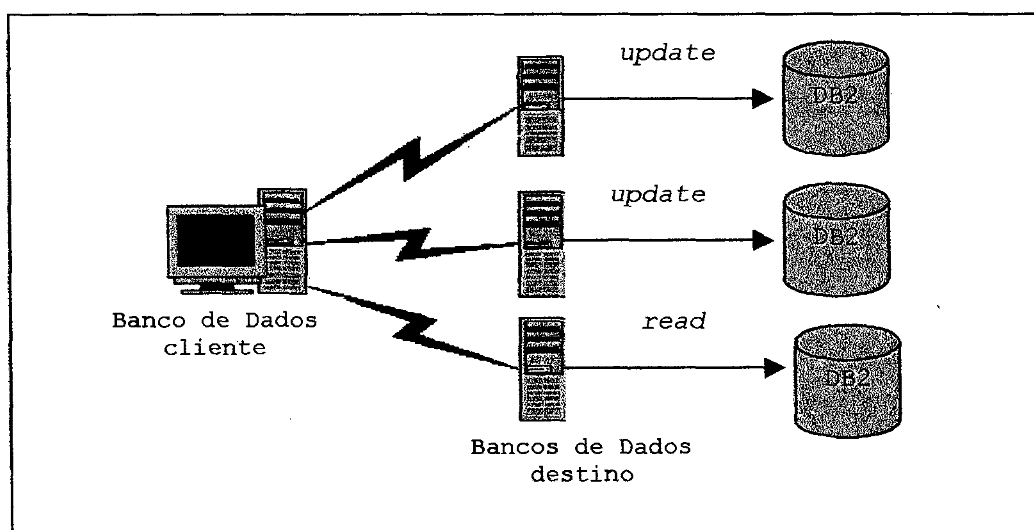

Figura 26: Atualizando Múltiplos Bancos de Dados em uma Simples Transação.

Os passos básicos para compor a estrutura descrita na figura 26 acima devem ser:

- Criar as tabelas necessárias nos bancos de dados apropriados que participam da distribuição;
- Se for fisicamente remoto, configurar nos servidores de banco de dados, os protocolos de comunicação para o uso apropriado, e ainda catalogar os nós e os banco de dados para identificação dos mesmos nos servidores;
- Pré-compilar as aplicações para especificar o tipo de conexão *on-phase commit ou two-phase commit*

- Configurar o DB2 *Transaction Mananger* (TM)

O gerenciador de banco de dados providencia as funções do gerenciador de transação, as quais podem ser usadas para coordenar a atualização de vários bancos de dados contendo uma UOW. O banco de dados cliente automaticamente coordena *unit of work*, e usa uma *transaction manager database* para registrar cada transação e o status completo da finalização de cada uma [IBM 2000c].

7.3 Comunicação e Conectividade do DB2 para Ambiente Distribuído

Como o DB2 possui portabilidade natural nas plataformas IBM, então os protocolos suportados para conexões entre DB2 *Universal Database* são: TCP/IP; SNA – *System Network Architecture*; NETBIOS; ou, IPX/SPX. Mas quando a conexão envolve plataformas como OS/390, VSE, VM ou *workstation* haverá a necessidade do uso de um produto IBM (*Connect Enterprise Edition*, ou *Connect Personal Edition)*.

Na conexão entre bancos de dados o *Connect* utiliza ferramentas de consultas, baseadas em *SQL* (Ex.: `SELECT Current SERVER FROM SYSibm.SYSdummy1`). No entanto há um conjunto de protocolos (DRDA – *Distributed Relational Database Architecture*) que permite à múltiplos sistemas de bancos de dados (IBM ou não IBM) que suas aplicações trabalhem juntas. Isto é, as combinações de protocolos destinadas ao gerenciamento de banco de dados relacional que usa o DRDA se conectam e formando um sistema de bancos de dados distribuídos. O DRDA (figura 27) coordena a comunicação entre os sistemas definindo quais e como as trocas devem ser realizadas [IBM 2000b].

Há varias combinações de modelos de aplicações (*3-tier, 2-tier, client server*) e protocolos de aplicações (ODBC, ADO, DB2 CLI, *Embedded* SQL, JDBC e SQLJ) que DB2 *Connect* suporta, além de uma completa infraestrutura de comunicação para interação com plataformas IBM e servidores de banco de dados DB2.

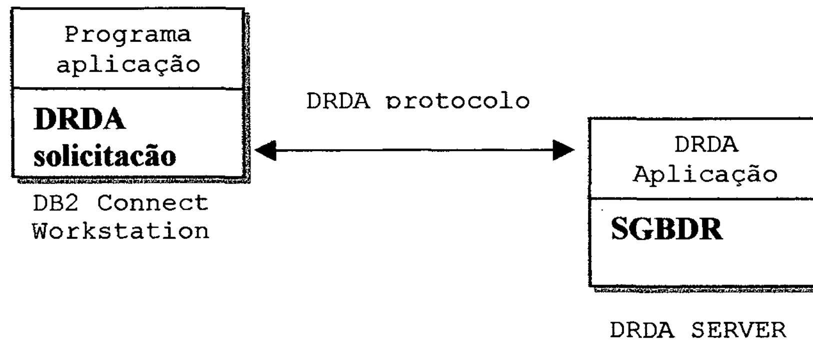

Figura 27: Troca de Dados entre Estação e Servidor de DRDA.

Ao implementar a arquitetura do DRDA, o DB2 *Connect* reduz o custo e a complexidade de acesso entre os dados armazenados no banco de dados DB2, principalmente quando a arquitetura é distribuída. Na implementação de conexões entre o sistema gerenciador de banco de dados e o DRDA os bancos de dados clientes usam as seguintes arquiteturas e protocolos na construção de blocos: CDRA – *Chracter Data Representation Architecture*; DDMA –*Distributed Data Management Architecture;* FD:OCA – *Formatted Data Object Content Architecture;* SNA – *system Network Architecture;* MAS – *SNA Management Services Architecture* e TCP/IP – *Transmisson Control Protocol/Internet Protocol*. Os dados que trafegam na rede são especificados pela arquitetura do DRDA, no qual o fluxo de dados suporta o acesso aos bancos de dados relacionais distribuídos [IBM 2000b].

Com o uso dessa arquitetura uma solicitação é roteada para destino correto por meios de diretórios que contém vários tipos de informações da comunicação, e o nome do servidor de banco de dados DRDA a ser acessado.

Acesso a grandes arquivos (dados de multimídia) pode gerar um enorme tráfico na rede, tornando a replicação ineficiente, considerando que esses dados não são atualizados com uma certa freqüência. Para essa situação o DB2 Universal providencia o DATALIK (disponível somente na versão 7 ou superior do DB2), um tipo de dado que permite ao DB2 o controle de acesso, integridade e recuperação dessas espécies de

dados. DB2 replica colunas do tipo DATALINK e usa o ASNDLCOPY como rotina de saída, para replicar arquivos externos para os quais as colunas DATALINK são utilizadas. Essa rotina transforma cada *link* origem em um *link* destino com referencia entre si, e copia arquivos externos do sistema de origem para o sistema destino.

7.4 Definições das Restrições de Integridade Distribuída no DB2

As definições das restrições de integridade naturalmente fornecidas pelo SGBD DB2 (*primary key, foreign key, unique e referential integrity*) são criadas pela DDL ou através dos comandos CREATE TABLE e ALTER TABLE da SQL (conforme o exemplo que segue). Porém há restrições quando a relação se destina à replicação (seção 7.1.5).

```
CREATE TABLE Curso
   (Codigo   INT(06)   NOT NULL,
   Nome      CHAR(30)  NOT NULL,
   PRIMARY KEY (Codigo))
```

É possível especificar condições onde são avaliadas cada tupla da relação através da verificação da restrição individual das colunas de uma relação. Isto é possível com uso de regras empregadas nas colunas para manter a integridade de seus valores através do *table check constraints,* conforme o próximo exemplo .

```
CREATE TABLE Curso (
      Codigo   INT(06)   NOT NULL,
      Nome   CHAR(30) NOT NULL,
      Valor   DEC (10,2) NOT NULL,
      PRIMARY KEY (Codigo))
      CONSTRAINTS Valor CHECK (Valor < 10000)
```

Da mesma maneira que é possível criar esse tipo de restrição, também pode ser alterada, no entanto a alteração pode violar a integridade dos dados existente. Mas com a opção SET (exemplo a seguir) é possível retardar a verificação da integridade quando ocorrer alteração.

```
SET CONSTRAINTS FOR Curso
ALTER TABLE Curso ADD CONSTRAINTS Valor
            CHECK (valor < 8000)
SET CONSTRAINTS FOR Curso IMMEDIATE CHECKED
```

Outra opção ainda mais relevante que pode reforçar a restrição de integridade é uso de *triggers*. Os *triggers* definem conjuntos de ações que são executadas ou disparadas, por uma operação de *INSERT, DELETE,* ou *UPDATE* em uma específica relação. Mas eles também podem escrever em outras relações, modificar entradas de valores e ainda disparar mensagens de alerta [IBM 2000d].

```
CREATE TRIGGER Aluno_Status
   NO CASADE BEFORE UPDATE OF Aluno ON
   Aluno.Status
   REFERENCING NEW AS Nstatus OLD as Ostatus
   FOR EACH ROW MODE DB2SQL
   ............
```

7.5 Cumprimento das Restrições de Integridade Distribuídas no DB2

O sistema gerenciador de banco de dados DB2 mantém as restrições de integridade de maneira geral, através das definições efetuadas no projeto físico do banco de dados. Mas essas definições são passíveis de alterações em função da necessidade dinâmica das organizações. Nesse sentido, os *triggers* desempenham papeis relevantes no cumprimento das restrições de integridade no DB2 destacando: validação da entrada de dados; geração automática de valores para um tupla que será inserida; ler outras relações com propósito de referencia cruzada; e, escrever em outras relações com proposta de auditoria [IBM 2000d]. Além disso, eles podem ser utilizados no desenvolvimento de aplicações que garantem de forma global as regras de negócio de uma organização.

Para o contexto distribuído do DB2 é indispensável o uso do *transaction mananger database* e do protocolo *two-phase commit,* principalmente para garantir a integridade das transações distribuídas e conseqüentemente as restrições de integridade definidas para esse modelo. Nesse sentido, alguns passos devem ser observados para o processamento do *two-phase commit* envolvido em um *multisite updade* [IBM 2000c]:

1- Quando o banco de dados cliente precisa conectar ao um outro banco de dados (para realizar qualquer tipo de operação), primeiro ele conecta ao *tansaction mananger database* (TM) internamente. O TM retorna a conexão ao banco de dados cliente que o reconhece.
2- A conexão para o banco de dados de destino é mantida.
3- Em seguida o banco de dados cliente inicia a atualização na relação(s) remota(s). O TM responde ao banco de dados cliente com ID para transação da *unit of work*. A *unit of work* ocorre somente quando o primeiro comando da SQL é executado e não quando se estabelece a conexão.
4- Após receber o ID da transação o banco de dados cliente o registra.
5- Outros comandos podem ser executados contra o(s) banco(s) de dados de destino(s) na mesma conexão, desde que a *unit of work* esteja registrada no banco de dados destino.
6- Quando o banco de dados cliente requer o *commit* da *unit of work*, uma mensagem "prepare" é enviada a todos os banco de dados participantes da *unit of work*. E cada banco de dados escreve "PREPARED" em seus registros *logs files*, e responde ao banco de dados cliente.
7- Após o banco de dados cliente receber uma resposta positiva de todos os bancos de dados, ela envia uma mensagem para TM, informando que a *unit of work* está preparada para ser *committed*. O TM escreve um "PREPARED" no seu registro *log file* e envia uma resposta informando ao banco de dados cliente que a segunda fase do processo de *commit* começou.
8- Durante a segunda fase do processo de *commit*, o banco de dados cliente envia uma mensagem a todos os bancos de dados participantes para receberem o *commit*. Cada banco de dados escreve "COMMITED" em seu registro do *log file*, e libera os *locks* que estavam presos na *unit of work*.
9- Antes do banco de dados cliente receber uma resposta positiva de todos os participantes, ele envia uma mensagem para o TM,

informando que a *unit of work* foi concluída. O TM então escreve "COMMITED" em seu registro do *log file,* indicando que a *unit of work* esta completa, e responde ao cliente indicando que ela terminou.

Os passos acima representam uma situação de normalidade, onde a execução do *two-phase commit* é finalizada com sucesso, garantindo a restrição de integridade do ambiente. Entretanto, a distribuição dos bancos de dados em vários servidores remotos incrementam potenciais erros como: falha na rede de comunicação e/ou servidor(es) inativo(s).

No entanto para o banco de dados DB2 o tratamento de erros durante o processo do *two-phase commit* também é abordado em duas fases:

- Primeira fase de erro:

  Se a comunicação do banco de dados falha no "PREPARE" do *commit* da *unit of work,* o banco de dados cliente fará o *roll back* da *unit of work* durante a segunda fase do processo de *commit.* Nesse caso, a mensagem "PREPARE" não é enviada ao TM. Durante a segunda fase, o cliente envia mensagem de "ROLLBACK" a todos os bancos de dados que tinha terminado com sucesso a primeira fase. Cada banco de dados escreve "ABORT" no registro do *log file* e libera os *locks* que estavam presos para *unit of work.*

- Segunda fase de erro:

  O tratamento de erro nessa fase depende de como a segunda fase do *two-phase commit* realizará o *commit* ou *roll back* da transação. A segunda fase fará o *roll back* da transação somente se for encontrado um erro na primeira fase do *two-phase commit.*

Na segunda fase de tratamento de erro o DB2 através do TM forçará a conclusão da transação distribuída mesmo que haja determinados tipos de erros. Por exemplo, Se um

dos bancos de dados participante do *commit* da *unit of work* falha, o TM forçará o *commit* nele. Nesse caso a aplicação será informada que o *commit* foi realizado com sucesso, pois o DB2 garante que a transação não *commited* no servidor seja *commited* através da configuração do *resyn_interval* (usado para especificar o gerenciamento de uma transação longa pelo servidor, o qual espera pela tentativa do *commit* da *unit of work*).

Se por alguma razão o TM não consegue resolver automaticamente as transações pendentes (*indoubt transaction*) , ou se decide não esperar pelo TM, então a solução deve ser manual. O processo manual muitas vezes é dirigido como uma decisão heurística [IBM 2000c].

Conclusão

A IBM por ser fabricante tradicional de hardware e software que mantém algumas características proprietárias de seus produtos. Por isso deve se estar atento ao projetar um ambiente distribuído para banco de dados onde envolvem plataformas IBM, pois há detalhes específicos entre seus produtos.

O DB2 possui algumas restrições na replicação de dados principalmente quanto ao cumprimento das restrições de integridade distribuída (seção 7.1.5). Quanto aos conflitos que podem ocorrer entre as transações durante a replicação de dados, o DB2 é radical, simplesmente detecta e desfaz as transações em conflito.

A distribuição de dados é totalmente gerenciada por unidades de trabalho (transações), mas essas transações se utilizam de um gerente que coordena as transações distribuídas. Quando não for possível o uso desse gerente, então o papel de coordenador fica por conta do próprio SGBD. Além disso, o DB2 disponibiliza o protocolo DRDA que permite qualquer SGBDR compatível ao protocolo compor um modelo distribuído de banco de dados. E ainda para ganhar agilidade nos acessos as bases distribuídas o DB2 utiliza um produto que é o DB2 *connect*.

Em meio a essa conectividade, a distribuição de dados no DB2 (replicação ou distribuição) se restringe apenas ao uso de *triggers* como mecanismo adicional no cumprimento das restrições de integridade distribuídas. Com isso é notória a ênfase ao protocolo *two-phase commit* para assistir a consistência dos dados distribuídos.

No próximo capitulo argumentaremos sobre as restrições de integridade distribuídas suportadas pelo SGBD SQL *server* 2000, bem como, as maneiras pelas quais o referido SGBD realiza a replicação e a distribuição de dados.

Capítulo 8

Restrições de Integridade Distribuídas no SQL 2000

A *Microsoft* lançou o SQL *server* no ano de 1988, em conjunto com *Asthon-Take Corporation* (legítimo desenvolvedor e distribuidor do *Dbase* nos anos 80) e a *Sybase* (*Sybase®Corporation* desenvolve e comercializa vários produtos de tecnologia de informação, o principal deles é o SGBD Sybase), na época o banco de dados da *Microsoft* foi motivo de "piada" entre os SGBD's que atuavam no mercado. Mas recentemente a *Microsoft® Corporation*[9] lançou a versão 2000 de seu SGBD, e essa versão faz frente aos principais SGBD's profissionais, além disso, a versão anterior (*SQL server* 7) abrigava uma boa parte do mercado de banco de dados [GUN 2001].

Este capítulo descreve o contexto distribuído para banco de dados suportado pelo sistema gerenciador de banco de dados *Microsoft* SQL *Server* 2000 (a partir desse ponto será referenciado como SQL 2000), bem como, a definição e o cumprimento das restrições de integridade distribuídas sustentadas pelo por ele.

8.1 Replicação de Dados no SQL 2000

A replicação no SQL 2000 é abordada por um conjunto de tecnologias que permite a manutenção de várias cópias de dados idênticas em múltiplos *sites*. No entanto o modelo de replicação (distribuição de dados) adotado pelo SQL 2000 é o *publisher-subscriber* composto pelo *publisher, subscriber e distributor:*

Um *publisher* é o servidor que origina os dados a serem replicados. Ele define um item para cada relação ou um outro objeto (*view, stored procedure)* do banco de dados para ser usado na origem da replicação. Um ou mais itens relacionados no mesmo banco de dados são organizados em uma publicação. Publicações são caminhos para grupos de

9: www.microsoft.com

dados e objetos que se deseja replicá-los [WEB 2001b].

O *subscriber* é um servidor que recebe os dados replicados pelo *publisher*. O *subscriber* define uma subscrição (espécie de assinatura) para uma particular publicação [WEB 2001b].

O *distributor* é um servidor que realiza varias tarefas, principalmente o de mediador quando os itens são movidos do *publishers* para *subscribers*. Essas tarefas dependem do tipo de replicação adotada.

O SQL 2000 realiza três tipos de replicação: *snapshot replication, transacional replication,* e *merger replication.*. Também suporta replicações originadas de dados heterogêneos através dos protocolos OLE DB ou ODBC que pode subscrever publicações no SQL *server*. O SQL 2000 também pode receber dados replicados de várias origens como: Microsoft Exchange, Microsoft Access, Oracle e DB2.

8.1.1 *Snapshot Replication*

A *snapshot replication* copia instantaneamente dados ou objetos do banco de dados exatamente como eles existem. As publicações *snapshot* são tipicamente definidas para acontecerem em bases programadas. Os *subscribers* contem cópias dos itens publicados a medida em que eles aparecem no último *snapshot*. A *Snapshot replication* é usada onde a origem do dado é relativamente estática.

A *snapshot replication* é implementada por dois agentes: *snapshot agent;* e, *distribution agent*. O *snapshot agent* prepara os arquivos *snapshot* contendo: esquema, dados das relações publicadas e objetos armazenados em arquivos *snapshot folder*. Em seguida registra a sincronização dos *jobs* no banco de dados de distribuição *(Distributor)*. Por *default* o *snapshot folder* é localizado no *distributor*, mas pode ser especificado em localizações alternadas em vez de usar a localização *defualt* [WEB 2001c]. O *distribution agent* move *snapshot* armazenado nas relações do banco de dados de distribuição, para as relações de destino nos *subscribers*. O banco de dados de

distribuição é usado somente pela replicação e não contém nenhuma relação definida pelo usuário.

A cada momento o *snapshot agent* é executado, ele verifica se qualquer nova subscrição foi adicionada. Se não há novas subscrições, nenhum script, ou arquivos de dados são criados. Se a publicação é gerada com a opção para criar imediatamente o primeiro *snapshot*, então novos esquemas e arquivos de dados são criados no momento em que o *snapshot agent* é executado. Todos os esquemas e arquivos de dados são armazenados no *snapshot folder*, portanto o *distribution agent* ou *merger agent* os transfere para *subscriber*. Essa transferência ainda pode ser efetuada manualmente. O *snapshop agent* realiza os seguintes passos [WEB 2001c]:

1- estabelece uma conexão do *distributor* para o *publisher* e prepara *share-locks* em todas as relações inclusas na publicação. O *share-lock* garante uma consistência *snapshot* de dados. Porque os *locks* não permitem que outros usuários atualizem as relações, o *snapshot agent* deve ser programado para executar quando a atividade do banco de dados estiver baixa.

2- estabelece uma conexão do *publisher* para o *distributor*, e escreve uma cópia do esquema da relação para cada item para arquivo (.sch). Se houver a inclusão de índices e declarativas de integridade referencial, o script do agente de saída seleciona os índices para um arquivo (.idx). Outros objetos do banco de dados, como *stored procedure*, *views*, funções definidas pelo usuário, entre outros, também podem ser publicados como parte da replicação.

3- copia os dados das relações publicadas no *publisher* e os escreve no *snpshopt folder*. Se todas os *subscribers* são instancias do SQL *server*, o *snapshot* é armazenado como volume nativo. Se um ou mais *subscribers* é uma origem de dados heterogêneos, o *snapshot* é armazenado em um arquivo no modo caractere.

4- anexa linhas para o `Msrepl_commands` e o `Msrepl_transactions` nas relações no banco de dados de distribuição. As entradas das relações `Msrepl_commands` são comandos indicando a localização do conjunto de sincronização (arquivos .sch e .bcp) e referencias para qualquer script de pré-criação especificada. As entradas nas relações `Msrepl_transactions` referenciam o *subscriber* da tarefa de sincronização.

5- libera os *share-locks* de cada relação publicada e finaliza escrevendo os históricos de *logs* na relação `Msrepl_transactions`

Por sua vez o *Distribution agent* realiza os seguintes passos [WEB 2001c]:

1- estabelece uma comunicação entre o servidor (em que o *agent* está localizado) e o *distributor*. Para mandar as subscrições, o *distribution agent* é executado normalmente no *distributor*, e para obter as subscrições o *distribution agent* é executado no *subscriber*.

2- examina as relações `Msrepl_commands` e `Msrepl_`transactions no banco de dados de distribuição no *distributor*. O agente aponta a localização da sincronização da primeira relação e os comando de sincronização do *subscriber* junto das tabelas.

3- aplica o esquema e comandos para a subscrição do banco de dados. Se o *subscriber* não é uma instância do SQL 2000, o agente converte os tipos de dados conforme a necessidade. Todos os itens de publicação são sincronizados, preservando a transação e a integridade referencial entre as relações bases.

A figura 28 mostra a arquitetura da replicação do tipo *snapshot*, na qual os agentes se cooperam para efetuar a replicação desse tipo.

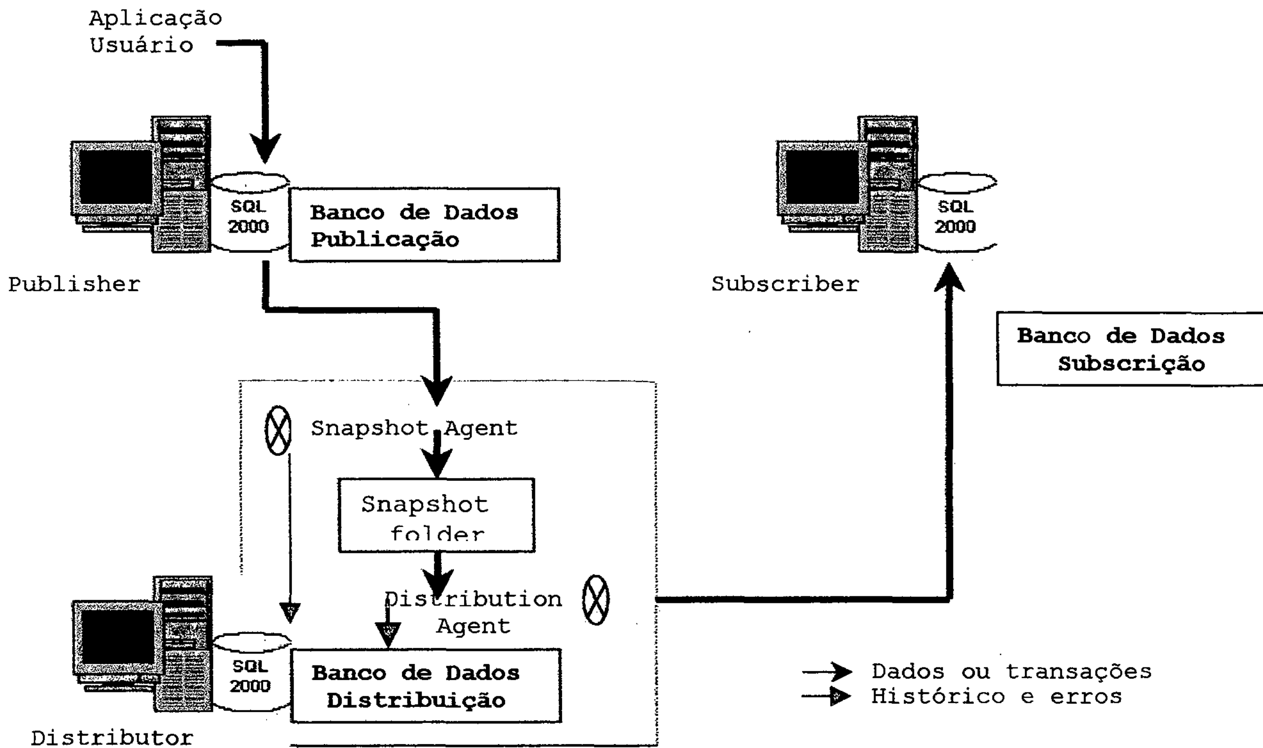

Figura 28: Arquitetura do *Snapshot Replication.*

8.1.2. *Transactional Replication*

Na *transactional replication* os *subscribers* são sincronizados primeiro com o *publisher,* tipicamente usando um *snapshot,* então as modificações de dados e transações são capturadas e enviadas para o *subscribers.* A integridade transacional é mantida nos *subscribers* por ter todas as modificações feitas no *publishers,* que são replicadas para *subscribers.* A replicação transacional é usada quando os dados devem ser replicados e modificados, sendo assim a transação deve ser preservada, os *publishers* e os *subscribers* são confiáveis, pois freqüentemente estão conectados em rede [WEB 2001d].

A *Transacional Replication* é implementada pelos seguintes agentes: *snapshot agent, log reader agent* e *distribution agent.* O *snapshot agent* prepara os arquivos contendo esquema, dados das relações publicadas e os objetos do banco de dados, armazena-os em um arquivo no *snapshot folder,* e ainda registra *jobs* de sincronização no banco de dados de distribuição no *distributor.*

O *log reader agent* monitora o *log* da transação para cada banco de dados configurado para replicação transacional. Copia as transações marcadas para a replicação do *log* de transação no banco de dados de distribuição. O *distribution agent* move os *jobs snapshot* e as transações seguras nas relações do banco de dados de distribuição para *subscribers* [WEB 2001d].

Esse tipo de replicação (figura 29) permite atualizações nas transações de dados marcadas para a replicação, então os mecanismos da *transactional replication* devem ser analisados com mais rigor devido a complexidade inerente a esse tipo de replicação.

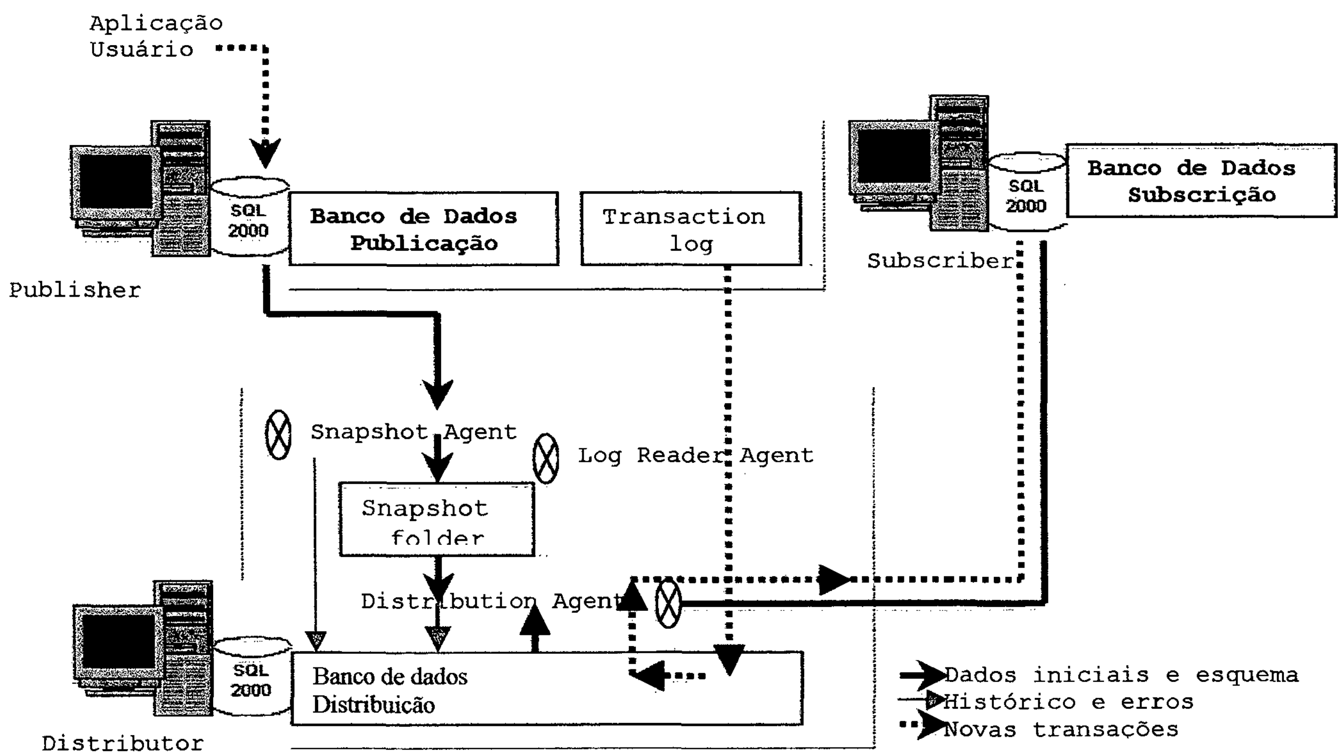

Figura 29: Arquitetura da *Transactional Replication*..

Antes do *subscriber* receber a *transactional replication* o *publisher* pode sofrer alterações incrementais, mas o *subscriber* deve conter as relações com o mesmo esquema e dados como os do *publisher*. A cópia completa da publicação corrente do *publisher* para o *subscriber* é chamada de aplicação do *initial snapshot*. Quando *snapshots* são distribuídos e aplicados nos *subscribers*, somente aqueles *subscribers* que

esperaram pelo *initial snapshots* são afetados, enquanto que outros *subscribers* não são [WEB 2001d].

Enquanto *snapshot* é gerado o SQL 2000 normalmente aplica *share-locks* em todas as relações publicadas como parte da replicação. Isto previne as atualizações das relações na publicação.

Concorrência com processamento *snapshot* é disponível somente com *trasactional replication*. Mas durante as entradas para geração do *snapshot* os *shrares-locks* não aparecem, permitindo aos usuários continuarem trabalhando criando seus arquivos *initial snapshot* normalmente.

8.1.3. *Merge Replication*

A *merge replication* permite a múltiplos *sites* trabalharem independente com um conjunto de *subscribers*, portanto uma fusão combinada de trabalho volta para *publisher*. Os *subscribers e publishers* são sincronizados como um *snapshot*. Modificações são rastreadas nos *subscribers* e *publishers*. Em algum momento depois as modificações são agrupadas para uma nova versão de dados. Durante a junção alguns conflitos podem ser encontrados quando múltiplos *subscribers* modificam o mesmo dado.

A *merger replication* suporta algumas definições para a resolução de conflitos, as quais são conjuntos de regras que define como resolver esses conflitos. *Custom conflict resolver script* pode ser escrito em qualquer lógica, o qual pode ser necessário para resolver conflitos considerando a complexidade do cenário do conflito. A *Merge replication* é usada onde um *subscriber* opera de forma independente, ou quando vários *subscribers* podem atualizar o mesmo dado [WEB 2001b].

A *Merge replication* é implementada por dois agentes: *snapshot agent* e *merger agent*. O *snapshot agent* prepara os arquivos *snapshot* contendo o esquema e os dados das relações publicadas, armazena os arquivos no *snapshot folder*, e insere a

sincronização dos *jobs* no banco de dados de publicação. O *snapshot agent* também cria replicações específicas, armazenando *procedures, triggers, e system tables* (relações do sistema) [WEB 2001e].

O *Merge agent* aplica o *initial snapshot jobs* nas relações do banco de dados de distribuição para os *subscribers.* Eles também agrupam as modificações incrementais de dados que ocorrem no *publisher* ou no *subscribers* depois que o *initial snapshot* tenha sido criado, e reconcilia os conflitos de acordo com as regras configuradas ou escritas no *custom resolver.*

O papel do *distributor é muito limitado* na *merger replication,* conseqüentemente a implementação do *distributor* é simples, pois o *distributor agent* não é usado durante toda a *merge replication.*

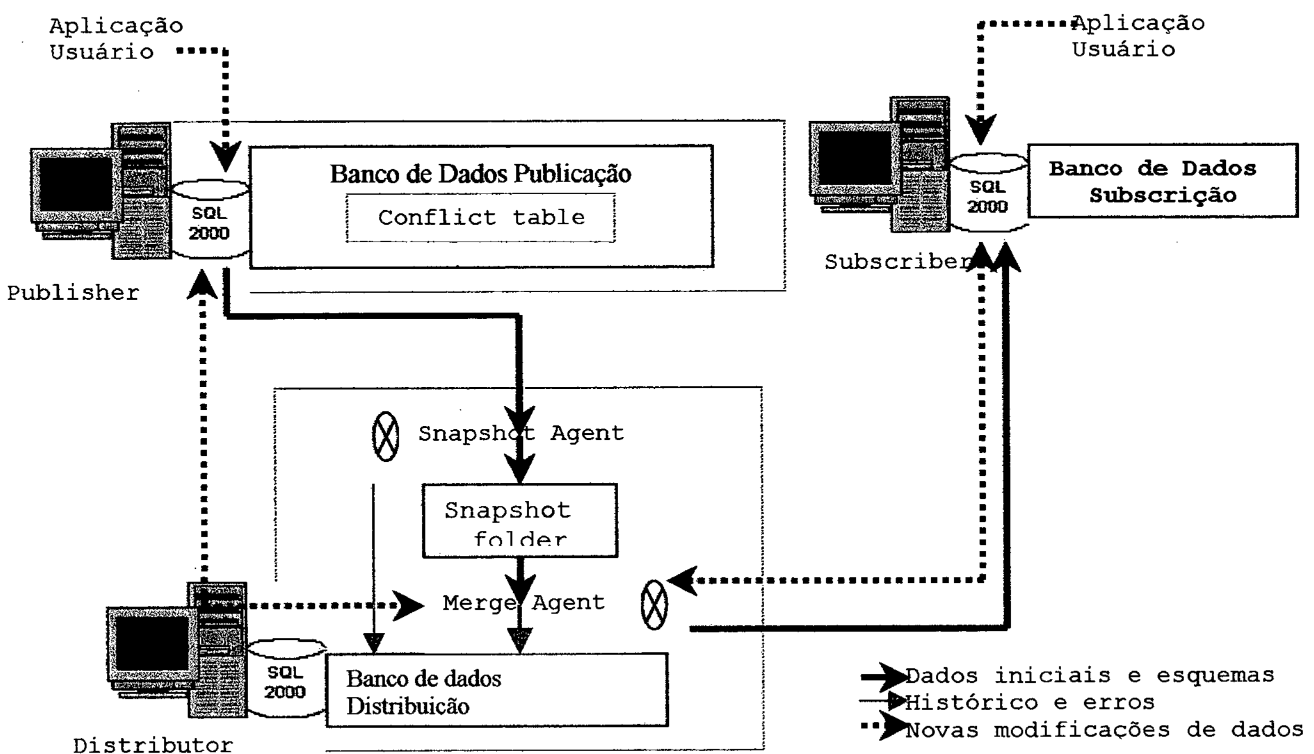

Figura 30: Arquitetura da *Merge Replication..*

A *Merge replication* conforme demonstrada na figura 30 possui uma complexidade mais elevada que os demais tipos de replicação (*transacional replication* e *snapshot replication*), ela envolve e suporta vários componentes como: *uniqueidentifier column; triggers; stored procedures; system tables,* além das suas próprias características.

Através do *uniqueidentifier column,* O SQL 2000 identifica uma única coluna para cada linha da relação replicada. Isso permite que a linha possa ser única em múltiplas cópias da relação. Se a relação contém uma coluna com a propriedade ROWGUIDCOL que possui um índice único ou a restrição de chave primária. O SQL 2000 usará essa coluna automaticamente como o identificador da linha para publicação da relação.

Os *triggers* quando utilizados na *merger replication* rastreia as modificações dos dados em cada linha ou em cada coluna, captura as modificações realizadas nas relações publicadas e registra as modificações ocorridas nas relações *merger system.* Diferentes *triggers* são gerados para rastrear os itens modificados no nível de coluna ou no nível de linha. Não há interferência de aplicações que utilizam *triggers* com a *merger replication.*

O *snapshot agent* também cria *stored procedured* customizadas que atualiza o *subscription database.* Também encontramos uma *stored procedured* customizada para os comandos de manutenção do *subscription database* (*insert, delete* e *update*). Quando o dado é atualizado ou e um novo registro precisa ser inserido no *subscription database,* então as *stored procedured* customizadas são usadas preferencialmente para os comandos (*insert, update* e *delete*) individualmente.

As *system tables* são adicionadas no banco de dados para suportar o rastreamento de dados, sincronização eficiente, e detecção de conflito. Para qualquer linha criada ou modificada na relação, a relação `Msmerge_contents` contém a criação das mais recentes modificações ocorridas. Ela também possui a versão da linha, assim como para qualquer atributo dessa linha. A relação `Msmerge_tombstone` armazena os *delete's* dos dados contidos em uma publicação [WEB 2001e].

8.1.4 Resolução de Conflitos e Sincronização no SQL 2000

A Sincronização ocorre quando *publishers* e *subscribers* em uma topologia *merge replication* se conectam e as modificações são propagadas entre os *sites*. Se necessário, os conflitos detectados são resolvidos. No momento da sincronização, o *merge agent* envia todas as modificações de dados para o *subscriber o* fluxo de dados do gerador da modificação para o *site* que necessita ser atualizado ou sincronizado[WEB 2001e].

A direção do controle de trocas será o *merge agent uploads changes* do *subscriber* (Exchange Type = 'Upload') , *downloads changes* para *publisher* (Exchange Type = 'Dowload') ou executar um *upload* seguido por um *download* (Exchange Type = 'Bidirectional').

No banco de dados de destino as atualizações propagadas de outros *sites* são reunidas com valores existentes de acordo com detecção de conflito, e regras de resolução. Um *merge agent* avalia a chegada e os valores dos dados correntes, e qualquer conflito entre os novos e velhos valores são resolvidos automaticamente baseado no *resolver default*. O *resolver* é especificado quando a publicação é criada ou um *custom resolver* que pode ser implementado de acordo com a lógica de negócio [WEB 2001e].

Os valores dos dados modificados são replicados para outros *sites*, e convergido com modificações feitas em outros *sites*, somente quando a sincronização ocorre. A sincronização pode ocorrer em minutos, dias ou uma vez por semana., ou seja, definido no *merge agent schedule* [WEB 2001e].

O período de retenção para a subscrição específica em cada controle de publicação é de acordo com a freqüência de sincronização entre o *publisher* e o *subscriber*. Se as subscrições não sincronizam com o *publisher* contendo o período de retenção, então elas são marcadas como *'expired'* e necessitará de ser reinicializado. Isto é, para evitar dados antigos no *subscriber* para sincronização e *uploading* dessas modificações para o *publisher*. O período de retenção *default* para a publicação é de quatorze dias. Porque o

merge agent enxuga as publicações e subscrições dos bancos de dados com base nesse período [WEB 2001e].

Na *merger replication* os *subscribers* atualizam dados baseado no original *snapshot,* a menos que se sinalize a subscrição para reinicialização. Quando a subscrição é marcada para a reinicialização, em seguida o *merger agent* é executado, ele aplicará um novo *snapshot* para o *subscriber*. Opcionalmente, as modificações feitas no *subscriber* são enviadas para o *publisher* antes que o *snapshot* seja reaplicado. Isso garante que qualquer modificação de dados no *subscriber* não será perdida quando a subscrição é reinicializada.

8.1.5 Opções de Replicações no SQL 2000

Há várias opções onde o projetista pode construir a replicação em conformidade com a necessidade da organização. Dentre elas as principais estão descrita na tabela 1.

Tabela 01: Opções de Replicação no SQL 2000.

Tipo de Replicação	Benefícios Proporcionados
Snapshot Replication *Transactional Replication* *Merge Replication* **Com a opção** *Filtering publised data*	Permite a criar partições horizontais e/ou verticais de dados que podem ser publicados como parte da replicação. Com a distribuição de porções de dados para diferentes *subscribers* com objetivo de: - Minimizar a quantidade de dados que são enviados sobre a rede. - Reduzir a quantidade de espaço de armazenamento no *subscriber*. - Customizar publicações e aplicações com base nos requisitos individuais dos *subscribers*. - Reduzir a incidência de conflitos porque diferentes porções de dados podem ser enviadas para diferentes *subscribers*.
Snapshot Replication *Transactional Replication*	Permite atualização de dados no *subscriber* e propaga essas atualizações para o *publisher* imediatamente, ou armazena na fila. A atualização das *subscripions* é melhor para a topologia de replicação onde os dados replicados geralmente são lidos, e

Com a opção *Updatable Subscriptions*	ocasionalmente atualizados no *subscriber*.. Quando *publisher, distributor* e *subscriber* estão conectados em maior tempo e quando conflitos causados por múltiplos usuários não são freqüentes.
Merge Replication **Com a opção** *Updatable Subscriptions*	Nessa opção a *merge replication* permite a atualizações de dados no *subscriber* ou no *publisher* e sincroniza as modificações de dados continuamente, seja sobre demanda ou em intervalos programados.
Snapshot Replication *Transactional Replication* **Com a opção** *Transforming Published Data*	Durante a movimentação dos dados há necessidade de transformar os dados publicados, ou seja, criar divisões para *snapshot* e *transacional* publicações. Transformando dados inicialmente publicados com tipos mapeáveis a outros tipos. Por exemplo, transformando dados inteiros em reais, concatenando dois ou mais campos em um, enfim manipulando *strings* e funções.
Merge Replication **Com a opção** *Optimizing Synchronization*	Para otimização da sincronização durante a *merge replication,* armazenar-se as informações no *publishers* em vez de transferi-las sobre rede ao *subscriber.* Isso garante uma certa performance na sincronização de conexão pesada na rede, mas requer a adicional armazenagem no *publisher*.

8.1.6. Comunicação nas Replicações com SQL 2000

A comunicação na replicação de dados no SQL 2000 é suportada pelo TCP/IP ou *multiprotocol network protocol.* O SQL 2000 usa TCP/IP *sockets* ou *multiprotocol NET-libraries* sobre o TCP/IP para estabelecer uma conexão ODBC entre o *publisher, distributor, subscribers* e outros. A sincronização entre os componentes da distribuição (*publisher, distributor* e *Publisher*) e agentes é efetuada através do *Internet Protocoll* (IP).

O TCP/IP *Sockets Net-Library* é habilitado por *default* na instalação do SQL 2000, porém quando a instalação é customizada esses serviços de comunicação podem estar indisponíveis. Em determinadas situações pode se especificar caminhos de FTP- *File*

Transfer Protocol e portas com um *snapshot folder location* sob as propriedades de publicação assim que um servidor estiver configurado como um *site* de FTP.

8.2. Distribuição de Dados no SQL 2000

Esta seção apresenta a estrutura de distribuição de dados suportada pelo SQL 2000, bem como o funcionamento dos principais mecanismos utilizados no gerenciamento e manutenção dos dados distribuídos entre vários bancos de dados SQL 2000, ou não.

O SQL 2000 oferece distribuição de dados através de dois principais mecanismos: transação distribuída, consulta distribuída e a junção de ambos. Há propósitos distintos entre eles, os quais serão apresentados a seguir.

8.2.1 Transações Distribuídas

Transações Distribuídas são transações que envolvem recursos de duas ou mais origens de dados. O SQL 2000 suporta transações distribuídas, permitindo a criação de transações que atualizam múltiplos bancos de dados SQL 2000 e outras origens de dados. Uma *distributed transaction* envolve dois componentes de software: *resource managers;* e, *transaction manager* [SQL 2000a]:

- *Resource managers* é um software que tem o controle de cada recurso envolvido em uma transação distribuída conhecido como gerente de recurso. Uma transação distribuída é composta na transação local em cada gerente de recurso individualmente. Cada gerente de recurso deve ser capaz de realizar *commit* ou *roll back* da transação local em coordenação com todos os outros gerentes de recursos na transação distribuída. O SQL 2000 pode operar como um gerente de recurso na transação distribuída em conformidade com a especificação do *X/Open XA* para o processamento de transação distribuída.

- *Transaction manager* realiza *committing* ou *rolling back* de uma transação distribuída controlada por um componente de software conhecido como gerente de transação. O gerente de transação em conjunto com cada gerente de recurso garante que todas as transações locais e a transação distribuída são *committed* ou *rolled back* juntas. O *Microsoft Distributed Transaction Coordinator* (MS DTC) *services* opera como um gerente de transação. MS DTC possui conformidades com a especificação do *X/Open XA* para o processamento de transação distribuída.

O Protocolo 2PC – *Two-phase commit* é o componente essencial da *distributed transaction* mas será abordado a seguir no item de cumprimento da restrição de integridade desse contexto.

Uma transação que contém uma simples instância do SQL 2000 que cobre dois ou mais banco de dados é atualmente uma transação distribuída. O SQL 2000, entretanto, gerencia internamente a transação distribuída; para o usuário ele opera como se fosse uma transação local.

O SQL 2000 participa de uma transação distribuída por: invocar *stored procedures* em servidores remotos que executam SQL 2000; automaticamente ou explicitamente promove uma transação local à uma transação distribuída e inscreve os servidores remotos na transação; efetua atualizações distribuídas que a atualiza dados em múltiplas origens de dados do tipo OLE DB (OLE DB é uma API usada para acessar dados é também recomendado para desenvolvimento de ferramentas, utilitários, ou componentes de baixo nível que necessita de alta performance); e se as origens de dados OLE DB suportam a interface OLE DB na transação distribuída, então o SQL 2000 também pode participar dessa transação distribuída [SQL 2000c].

O serviço do MS DTC é coordenar a finalização da transação distribuída para garantir que todas as atualizações em todos os servidores são efetuadas permanentemente, ou em caso de erros todas são apagadas. A figura 31 evidencia essa

coordenação do MS DTC quando um cliente efetua uma chamada remota de *stored procedures.*

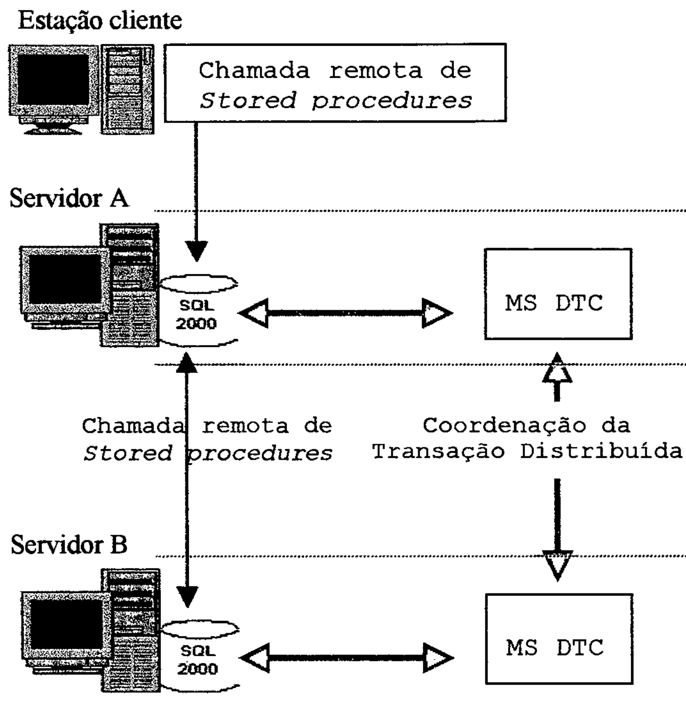

Figura 31: Transação Distribuída com Coordenação Própria no MS DTC.

As aplicações em SQL 2000 também podem chamar diretamente o MS DTC para iniciar explicitamente a transação distribuída. Um ou mais servidores executando SQL 2000 podem ser instruídos a participarem da transação distribuída e coordenar a própria finalização da transação com o MS DTC conforme a figura 32 [SQL 2000d].

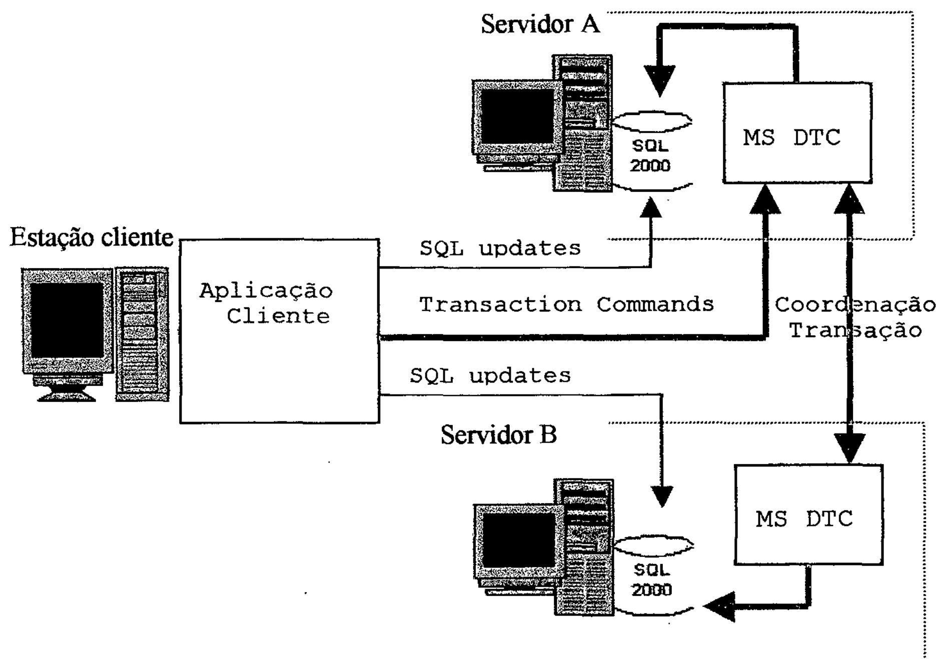

Figura 32: Aplicações de SQL 2000 Invocando Diretamente Transação Distribuída.

A transação distribuída deve ser atômica; isto é, ela deve terminar em todos gerentes de recursos ou eles terminarem ela. A aplicação do cliente para suportar o MS DTC deve utilizar o *flag* de transação `TXN_USE_DTC`, e os métodos `GetOption` e `SetOption` para o flag da transação, os quais devem ser suportados no `IrepositoryTransaction`. As aplicações podem ser mais fáceis de serem implementadas nas linguagens da *Microsoft* como *Visual C++ e Visual Basic.*

8.2.2. Consultas Distribuídas

As consultas distribuídas acessam dados de múltiplas origens heterogêneas, as quais podem estar armazenadas no mesmo servidor ou em diferentes servidores. O SQL 2000 suporta consultas distribuídas em função do uso do OLE DB – *object linking and embedding database*, e especificação de API – *application programming interface* para o acesso universal de dados.

A Consulta Distribuída proporciona o acesso aos usuários do SQL 2000 nas seguintes situações:

- Dados distribuídos armazenados em múltiplas instancias de SQL 2000.
- Dados heterogêneos armazenados em várias origens relacionais ou não relacionais através do OLE DB.

O OLE DB provê a exposição de dados em objetos tabular chamados *rowsets*. O SQL 2000 permite que as *rowsets* fornecidas pelo OLE DB possam ser referenciadas por comandos do *Transact-SQL* (comandos da SQL com transações embutidas. também utilizado para construção transações distribuídas) se elas forem uma relação do SQL 2000 [SQL 2000b].

Relações e *views* em origens externas de dados podem ser referenciadas diretamente pelos comandos do *Transact-SQL* (`INSERT`, `SELECT`, `UPDATE`, e `DELETE`), porque *distributed queries* usa OLE DB como interface base, *distributed queries* podem acessar os tradicionais SGBD's com processador de consultas SQL. Então os dados de sistemas proprietários expostos em uma tabular *rowset* através do OLE DB *provider* podem ser usados na *distributed queries*. Porém uma questão deve ser observada, ao usar a consulta distribuída no SQL 2000, ela é similar à funcionalidade do *linked table* (apresentado a seguir como *lnked server*) através do ODBC – *open data base connectivity*. Esta funcionalidade agora é construída no SQL 2000 usando OLE DB como interface externa de dados.

Linked server é um servidor virtual que pode ser definido para o SQL 2000 contendo todas as informações necessárias para acessar origens de dados OLE DB. Com a configuração do *linked server* o SQL 2000 é capaz de executar comandos contra origens de dados OLE DB em diferentes servidores. Algumas das vantagens do *linked server*: acesso remoto ao servidor; habilidade para lançar consultas distribuídas, atualizações, comandos, e transações nas origens de dados heterogêneas; e, habilidade para apontar a similaridade entre as diversas origens de dados [SQL 2000d].

Um *linked server* descreve a definição de um OLE DB *provider* e de um OLE DB *data source.* Um OLE DB *provider* é uma DLL – *Dynamic-link Library* que gerencia e interage com uma origem de dados específica. Um OLE DB *data source* identifica a acessibilidade específica do banco de dados através do OLE DB. Embora as origens de dados consultadas através das definições do *linked server* usualmente são bancos de dados. O OLE DB fornece uma variedade de arquivos e formatos, incluindo arquivos textos, dados de planilhas eletrônicas e resultados de buscas em conteúdos de *full-text* [SQL 2000d].

A figura 33 apresenta a interconectividade do SQL 2000 através da interface base do OLE DB, e também com uso do protocolo ODBC, além do *linked Server* definindo OLE DB *provider* e o OLE DB *data source*.

Apesar da conectividade ser ampliada com OLE DB e o ODBC, o comportamento das consultas distribuídas contra uma relação remota depende da funcionalidade do OLE DB *provider* usada para acessar essa relação. A especificação do OLE DB define um conjunto de objetos para OLE DB *provider,* e cada objeto possui um conjunto de interfaces. Muitos desses objetos e interfaces são opcionais e pode não ser suportado por um *provider.* Se um OLE DB *provider* não suportar alguns desses objetos ou interfaces, a funcionalidade da consultas distribuídas que depende desses componentes não realizará o acesso remoto às relações através do provedor OLE DB [SQL 2000b].

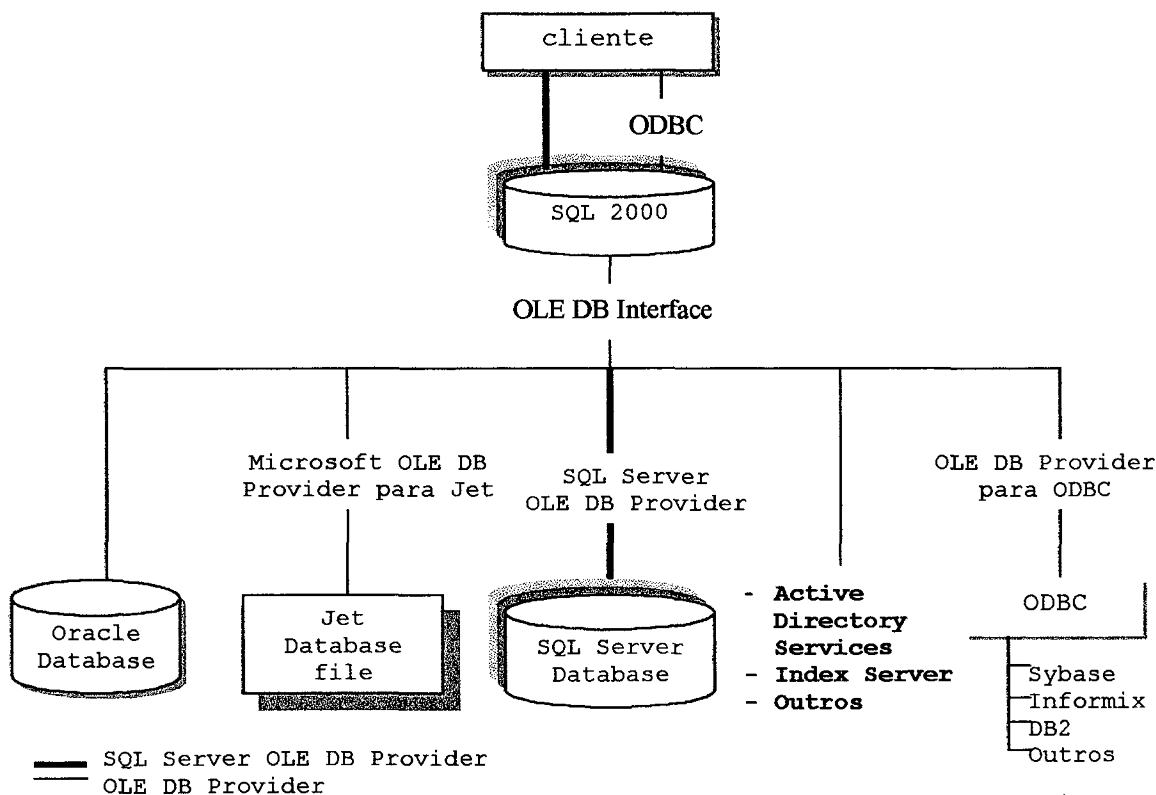

Figura 33: Estrutura Básica do Acesso à Dados Pela Consulta Distribuída.

8.2.3. Junção da Consulta Distribuída e da Transação Distribuída.

O SQL 2000 permite criar *links* para OLE DB *data sources* chamando de *linked servers* (descrito na seção 8.2.2). Após o estabelecimento do *link* a um OLE DB *data source* é possível referenciar *rowsets* do OLE DB *data sources* como relações nos comandos *Transact-SQL*, e passar comandos para OLE DB *data sources* e incluir o resultado das *rowsets* como relações do *Transact-SQL*.

Cada consulta distribuída pode referenciar múltiplos *linked servers* e efetuar operações de atualização ou leitura contra cada individual *link server*. Uma simples consulta distribuída pode realizar operações de leitura em alguns *linked servers* e operações de atualização em outros *linked server*. Em geral o SQL 2000 requer o suporte transações distribuídas do provedor OLE DB correspondente quando os dados surgem em mais de um *linked server* que provavelmente serão atualizados na transação. Portanto, os tipos de consultas abordadas contra *linked servers* dependem do nível de suporte para as transações presentes nos provedores OLE DB. Com tudo, o OLE DB

define duas opções de interfaces para o gerenciamento da transação: `ITransactionLocal` que suporta transações locais no OLE DB *data source;* e, `ItransactionJoin` que provê a junção a uma transação distribuída incluindo outros gerentes de recursos [SQL 2000b].

Qualquer OLE DB *provider* que suporta `ITransactionJoin` também suporta `ITransactionLocal`. Se uma *distributed query* é executada quando a conexão é *autocommit,* as seguintes regras são aplicadas: somente operações de leituras são permitidas contra os provedores que não suportam `ITransactionLocal` e, todas as operações de atualização são permitidas contra qualquer provedor que suporta `ITransactionLocal`.

Para o SQL 2000 o controle das chamadas para `ITransactionLocal` é automático para cada *linked server* participante de uma operação de atualização, assim como os *commits* e *rollbacks.*

Se uma consulta distribuída é executada contra uma *view* de particionamento distribuída, ou se ela é executada quando a conexão da transação é explicita ou implícita, as seguintes regras são aplicadas: somente operações de leitura são permitidas contra os provedores que não suportarem `ItransactionJoin`; provedores que não suportam qualquer transação ou suporta somente `ITransactionLocal` não pode participar de uma operação de atualização; Se `SET XACT_ABORT` é `ON` todas as operações são permitidas contra qualquer provedor que suportar o `ITransactionJoin;` e, Se `SET XACT_ABORT` é `OFF`, o *linked server* pode suportar também transações aninhadas antes das operações de atualizações serem permitidas [SQL 2000b].

8.2.4 Comunicação e Segurança na Distribuição de Dados no SQL 2000.

A comunicação no SQL 2000 é suportada pela *Net-Libraries* uma espécie de DLL como um particular protocolo de rede. O par de compatibilidades da *Net-Libraries* deve ser ativado no cliente e no servidor para suportar o protocolo desejado da rede.

As instancias de SQL 2000 suporta somente alguns protocolos como: *named pipes; TCP/IP sockets; NWLink IPX/SPX;* e, *shared memory net-libraries*. No entanto não suportam multiprotocolos como *AppleTalk*, ou *Banyan VINES*.

Nas transações dependentes de comunicação há uma opção que torna a conexão mais robusta com uso da *bound connections*. *Bound connections* permite duas ou mais conexões compartilharem a mesma transação e *locks*. *Bound connections* pode agir no mesmo dado sem o conflito de *lock*. Ele pode ser criado por múltiplas conexões com a mesma aplicação, ou por múltiplas aplicações com conexões separadas. *Bound connections* faz a coordenação das ações cruzadas em múltiplas conexões [SQL 2000b].

Para participar do *bound connections*, uma conexão chama `sp_getbindtoken` ou `sev_getbindtoken` e toma um *bind token*. Um *bind token* é uma string de caracteres que identifica unicamente cada *bound transaction*. O *bind token* é enviado a outras conexões participantes da *bound connections*. As outras conexões *bind* na transação por chamar `sp_bindsession`, usando o *bind token* recebido da primeira conexão. O *bind tokens* deve ser transmitido no código da aplicação que fez a primeira conexão para o código da aplicação que fizer qualquer subseqüente *bounds connections*.

Somente uma conexão em um conjunto de *bound connections* pode ser ativada a qualquer momento. Se uma conexão está executando um comando no servidor ou esta aguardando um resultado pendente do servidor, nenhuma outra conexão que compartilha a mesma transação poderá acessar o servidor enquanto a conexão corrente não finaliza o processo ou cancela o comando. Se o servidor estiver ocupado, um erro ocorre indicando que espaço da transação está sendo usado e deverá tentar em outra oportunidade [SQL 2000b].

Há dois tipos de *bound connections: local bound connection* que permite compartilhar o espaço de uma transação simples em um servidor; *distributed bound connection* permite compartilhar a mesma transação em dois ou mais servidores enquanto que toda a transação é *commited* ou *rolled back* por usar MS DTC.

Bound connections distribuídas não são identificadas por uma string de caracteres *bind token*, elas são identificadas pelos números das transações distribuídas.

Para que um usuário se conecte a múltiplos servidores ele deve ter a credencial para autenticar seu *login* original através do *security account delegation*. Para o uso da delegação, todos os servidores que estão conectados devem habilitar o suporte para *Kerberos* e usar o produto *Microsoft Active Directory*. Esses produtos são compatíveis somente com estações *Windows* 2000 (sistema operacional da *Microsoft)*.

Durante as conexões entre *linked server* (processamento de uma consulta distribuída) *logins* e *password* são mapeadas e adicionadas pelo `sp_addlinkedsrvlogin` e removidas pelo `sp_droplinkedsrvlogin` (quando necessário). Um *linked server login* mapeado estabelece um remoto *login* e uma remota *password* para qualquer *linked server* e *login* local. Quando ocorre a conexão do SQL 2000 com *linked server* para executar uma consulta distribuída ou *stored procedure*, ele verifica qualquer mapeamento de *login* e também o corrente *login* que está executando o procedimento da consulta. Se não for um deles, ele envia *login* e *password* correspondente enquanto estiver conectado com *linked server* [SQL 2000d].

8.2.5 Restrições do *Transact-SQL* em Consultas Distribuídas

Na execução de uma consulta distribuída surgem algumas restrições para Transact-SQL como:

- Todas as consultas com a forma padrão do `SELECT` são permitidas, porém na clausula `INTO` *`new_table_name`* do `SELECT` não é permitida quando *`new_table_name`* refere a uma relação remota;
- Quando especifica um LOB – *large objet* a coluna de uma relação remota, a clausula `ORDER BY` não será admitida no `SELECT;`
- Os predicados `IS NULL` e `IS NOT NULL` não podem ser referenciados como uma coluna LOB em uma relação remota;

- GROUP BY ALL não é permitido em consulta distribuída quando a consulta possui a clausula WHERE . No entanto, GROUP BY sem ALL é permitido;
- Colunas com grandes tipos de objetos (como texto ou imagem) não podem ser referenciados em operações de atualizações ou inserções, se o provedor for um processo instantâneo fora do SQL 2000 [SQL 2000b].

8.3 Definições das Restrições de Integridade Distribuídas no SQL 2000

As definições das restrições de integridade no SQL 2000 naturalmente surgem nas cláusulas dos comandos CREATE TABLE e ALTER TABLE, os quais define as restrições básicas (chave primária, chave única, integridade referencial e domínio). No entanto, há algumas características de integridade que podem ser definidas com as restrições básicas, porém incrementadas. Por exemplo, restrição de integridade referencial em cascata, ou seja, uma deleção ou uma atualização pode ser acionada em cascata quando o valor de uma tupla é eliminado ou alterado, e esse valor constitui referencias em outras relações. A opção CASCADE deve surgir na cláusula REFERENCES conforme exemplo a seguir.

```
CREATE TABLE mytable
[ FOREIGN KEY ]
     REFERENCES ref_table [ ( ref_column ) ]
     [ ON DELETE { CASCADE | NO ACTION } ]
     [ ON UPDATE { CASCADE | NO ACTION } ]
```

Outra situação é a definição da cláusula CHECK constraints com o papel de garantir a integridade de domínios, escalas de valores válidos, valores boleanos e expressões lógicas. Existe a possibilidade de aplicar múltiplos CHECKs constraints a uma simples coluna. E também aplicar CHECK constraints a múltiplas colunas por ser criado em nível de relação [SQL 2000e].

```
CREATE TABLE mytable
        .....
CHECK [ NOT FOR REPLICATION ]
    ( search_conditions) ou
    (logical_expression)
```

Um mecanismo bastante eficiente que suplementa a integridade no SQL 2000 é o *trigger*. Ele pode acessar outras relações, incluir complexos comandos do *Transact-SQL* e ainda ter características próprias. Normalmente um *trigger* é invocado em resposta aos comandos de INSERT, UPDATE ou DELETE. No SQL 2000 *triggers* podem possuir características aninhadas de até 32 níveis, mas somente para as relações que não possuem restrições de integridade definida pela cláusula CASCADE. segue um exemplo de *trigger* no SQL 2000.

```
CREATE TRIGGER nduplica_aluno ON Aluno
   INSTEAD OF INSERT
   AS
   BEGIN
   SET NOCOUNT ON
            -- Verifica a duplicação de aluno.
   IF (NOT EXISTS (SELECT A.SSN
       FROM Aluno A, inserted I
       WHERE A.SSN = I.SSN))
       INSERT INTO Aluno
       SELECT *
       FROM inserted
       ELSE.......
```

8.4 Cumprimento das Restrições de Integridade Distribuídas no SQL 2000

O SQL 2000 trata a distribuição de dados como uma transação distribuída, conseqüentemente o gerenciamento dessas transações implica no cumprimento da restrição de integridade, principalmente para aquelas definidas no projeto, mas também se responsabiliza pela atomicidade da própria transação.

Um processamento de *commit* especial é requerido no *two-phase commit* para prevenir problemas no gerenciamento de transações que transpõem múltiplos gerentes

de recursos. Um *commit* de uma grande transação pode tomar muito tempo enquanto os *logs de buffers* são liberados. O próprio processo de *commit* pode encontrar erros o qual deve forçar um *rollback*. Se o gerente de transação já sinalizou para cada gerente de recurso o *commit*, então ele obtém de volta os status de sucesso de alguns gerentes de recursos, mas também pode obter erros de um dos gerentes recurso. Isto cria um conflito porque todas as transações distribuídas devem ser desfeitas, mas parte delas já terminaram. Sendo assim, o 2PC realiza em duas fases (*prepare e commit*) o *commit* para evitar esse tipo de problema [SQL 2000a].

- Na *prepare phase* o gerente de transação envia uma mensagem para cada gerente de recurso preparando-os para o *commit*. Então cada gerente de recurso realiza um esforço para completar o processo de *commit*, como a liberação de todos os *buffers* dos *logs*. O gerente de recurso retém somente o mínimo de *locks* necessários pata manter a integridade da transação, e então retorna o status com sucesso ou falha ao gerente de transação.

- Na *commit phase* se todos os gerentes de recurso retornam o status com sucesso para as solicitações do *prepare*, então o gerente de transação envia comandos de *commit* para cada gerente de recurso. Cada gerente de recurso rapidamente registra que a transação foi completada e libera os últimos recursos presos. Se qualquer gerente de recurso retorna um erro para a solicitação do *prepare*, o gerente de transação então envia comandos de *rollback* para cada gerente de recurso.

Do ponto de vista da aplicação, uma transação distribuída é gerenciada com a mesma intensidade como uma transação local. No fim da transação, a aplicação solicita a transação para ser *committed* ou *rolled back*. Um *commit distributed* pode ser gerenciado diferentemente por um gerente de transação minimizando o risco de falhas da rede, podendo resultar que em alguns gerentes de recursos o sucesso do *commited* enquanto outros desfazem a transação [SQL 2000b].

Evidentemente o protocolo que a coordena a atomicidade da transação em múltiplos gerentes de recursos é o *two-phase commit*. E no ambiente *Microsoft* há uma facilidade

de execução com o MS DTC. O *Microsoft Distributed Transaction Coordinator* é um gerente de transação que permite às aplicações cliente incluírem várias e diferentes origens de dados em uma transação. O MS DTC coordena a finalização da transação distribuída em todos os servidores listados na transação.

É notório que o cumprimento da restrição de integridade é de responsabilidade do mecanismo que gerencia transação principalmente distribuída. Mas algumas restrições definidas naturalmente na concepção das relações não devem ser aplicadas no contexto distribuído. Por exemplo, quando há replicação de um banco de dados em um *site* remoto a opção do `CHECK constraints` não deve ser utilizada mais, pois a integridade é ativada quando há entradas no banco de dados de origem e ainda a replicação pode falhar se houver violação de dados no `CHECK constraints` [SQL 2000f].

Quanto ao uso de *triggers* no cumprimento das restrições de integridade distribuídas, primeiramente ele deve obedecer aos mecanismos de gerenciamento de transações distribuídas. Deve ser projetado para suportar o ambiente distribuído (particionamento e replicação de relações em diversos servidores) e suas principais contribuições para o cumprimento da restrição de integridade distribuída são: estender a integridade referencial entre relações; inserir ou atualizar dados em relações e *views*; verificar erros e empregar a ação baseada no erro; e, encontrar diferenças entre o estado inicial e final da relação quando houver ação de modificação [SQL 2000g].

Conclusão

O SQL 2000 suporta várias combinações de distribuição de dados, mas é dependente de alguns componentes de software para gerenciar as transações distribuídas. No entanto, possui uma interoperabilidade bastante relevante, pois faz uso de vários protocolos destinados a heterogeneidade. A implementação de qualquer ambiente distribuído no SQL 2000 envolve a combinação de vários componentes de softwares alçando um nível de detalhes bastante relevante.

Existem algumas restrições para o *Transact-SQL* no ambiente distribuído principalmente para as consultas distribuídas (seção 8.2.5), esse fato pode implicar em alguns recursos adicionais para extrações específicas de dados distribuídos.

Para o problema principal da replicação de dados, o conflito entre transações, o SQL 2000 permite ao DBA "customizar" a resolução de conflitos (manual) e apenas resolve conflitos de maneira automática, com base nos valores correntes e valores antigos registrados na sincronização. Para esse o problema o SQL 2000 não oferece muitas opções de contorno, e na dúvida é melhor recuar as transações em conflitos, embora essa atitude possa causar um certo desconforto para aplicação, mas de alguma maneira garante a integridade dos dados distribuídos.

Quanto à definição das restrições de integridade estruturais elas são transparentes em relação ao ambiente (centralizado ou distribuído). O SQL permite o uso *triggers*, chamadas a procedimentos remotos para definir e também garantir algumas das restrições de integridade distribuídas. No entanto, são desabilitados a restrição de integridade referencial e o *check constraints* durante a replicação de dados (com o objetivo de realizar de qualquer forma a replicação dos dados). Além disso, é dependente do gerenciamento de transações distribuídas, sobretudo do protocolo *two-phase commit*. Sendo assim cabe a atenção redobrada ao projetar uma replicação de dados no SQL 2000.

No capítulo seguinte faremos uma análise comparativa entre os SGBD's abordados até o momento, tomando como base o contexto de replicação e distribuição de dados mencionados nos capítulos anteriores.

Capítulo 9

Comparações dos SGBDs Para o Modelo Distribuído

Com base na documentação técnica dos fabricantes dos SGBD's (Ingres, Oracle, DB2 e SQL 2000), a qual proporcionou a descrição sintética da replicação, distribuição, definição e cumprimento das regras de integridade distribuídas (capítulos 5, 6, 7 e 8), efetua-se neste capítulo um comparativo dos modelos de replicação e distribuição de dados, considerando as definições e cumprimento das restrições de integridade distribuídas discutidas anteriormente.

9.1 Avaliação Geral

Para os SGBD's a replicação é tratada e implementada de forma distinta em relação à distribuição, embora ambas contemplem o ambiente distribuído de cada SGBD. No entanto, boa parte dos principais mecanismos são utilizados tanto na replicação quanto na distribuição.

Quando se fala em bancos de dados distribuídos, a principal característica que deve ser avaliada é o suporte a fragmentação de dados (horizontal e/ou vertical), e em seguida se o SGBD possui mecanismos que mantém a integridade dos fragmentos de dados distribuídos. Neste sentido, apresenta-se a tabela 02 que demonstra a comparação entre os SGBD's considerando-se os itens indispensáveis ao contexto distribuído, sobretudo ao cumprimento das restrições de integridade (estruturais e comportamentais) distribuídas.

Tabela 02: Comparação entre os SGBD's para Modelo Distribuído de Banco de Dados

Elementos relevantes ao contexto distribuído de banco de dados.	**SGDB Ingres**	**SGBD Oracle**	**SGBD DB2**	**SGBD SQL 2000**
Replicação (réplicas de relações)	➢ +	➢ +	➢ +	➢ +
Fragmentação Horizontal (particionamento de tuplas)	➢ +	➢ +	➢ +	➢ +
Fragmentação Vertical (particionamento de atributos)	➢ -	➢ -	➢ -	*
Uso de Assertivas Distribuídas	➢ +	➢ +	*	➢ +
Uso de Linguagens de Programação para Suplementar as Restrições de Integridade	➢ +	➢ +	X	➢ +
Garantia do Cumprimento das RI's Através das Transações Distribuídas	➢ +	➢ +	➢ +	➢ +
Uso de *Triggers* como Mecanismo Auxiliar no Cumprimento das RI's	➢ +	➢ -	➢ +	➢ -

Legenda:
➢ é suportado
\+ sem restrições
\- com restrições
* não é suportado
x não menciona

9.2 Comparação na Replicação de Dados

Todos os SGBD's investigados suportam replicação de dados. Embora a parametrização dos mecanismos de alguns SGBD's demande maior complexidade, enquanto para outros a complexidade é menor. Há duas situações relevantes na implementação desse contexto - o tratamento de colisão e a sincronização entre as bases. Não havendo sincronia entre os *sites* também não haverá replicação, além disso, as transações que manipulam algum tipo de dado replicado dependem da sincronização para garantir sua atomicidade, e conseqüentemente a integridade dos dados replicados.

Quanto à resolução de conflitos (a possibilidade de conflito está diretamente ligada ao projeto de replicação abordado), ela deve ser administrada eficientemente através de mecanismos (automáticos e/ou manuais) fornecidos pelo SGBD, pois boa parte da consistência dos dados replicados dependem desse tratamento.

As tabelas 03 e 04 apresentam as características de cada SGBD para o contexto de replicação não determinando a eficiência entre eles, pois a eficiência depende da replicação projetada.

Tabela 03: Características Essenciais na Replicação de Dados.

SGBD	Uso de Artefatos de Hardware e Software adicional além do ambiente de rede	Sincronização e Seguraça	Resolução de Conflitos
Ingres	SIM	Sincronização é coordenada. Possui um excelente nível de segurança com certificação.	Manual e automática. A automática possui limitações.
Oracle	NÃO	A sincronização é transparente. A seguraça é bastante explorada através de protocolos de encriptação de dados.	Manual e automática. A automática é satisfatória.
DB2	SIM	Sincronização é coordenada. Segurança de acesso suportada apenas pelas plataformas IBM ou Microsoft windows 2000.	Somente manual.
SQL 2000	SIM	Sincronização programada. Segurança do ambiente Microsoft com *Kerberos.*	Manual e automática. A automática é configurada no *merger agent* com algumas restrições .

Tabela 04: Opções de Replicações e Restrições de Integridade.

SGBD	Opções de Replicações Suportadas	Restrições de integridade	Outras restrições
Ingres	Cinco combinações diferentes.	Ingres/*Replicator* utiliza o *two-fase commit protocoll.* Permite criar política de regras particulares através do *Dbevent,* usar *procedures* e aplicações na linguagem C com SQL embutido.	O Ingres/*Replicator* atualiza somente duas bases de dados por vez, a base local e base destino.

Oracle	Quatro opções diferentes.	Protocolo *two-phase commit* executado transparentemente nas transações. Uso de *triggers* e *Stored Procedures.*	As Transações não manipulam replicação de objetos em grupos em diferentes conexões. Na restrição de integridade referencial, não é garantido que o processo de deleção será efetuado na ordem correta, ou seja, `ON DELETE CASCADE` não é suportado. Não permite a atualização de valores das colunas que edificam as relações.
DB2	Possui cinco métodos com quatro cenários distintos.	As unidades de trabalho distribuídas (transações distribuídas) fazem uso do protocolo *two-phase commit.* E apenas o uso *Triggers.*	Nome de relações e usuários na replicação suporta apenas 18 caracteres; Dados compactados pelo EDITPROC ou FIELDPROC não são replicados; DB2 *DataPropagator* não captura chamadas para *stored procedures;* DB2 *DataPropagator* não suporta as seguintes palavras chaves da SQL : `CRETE TABLE` para replica de relações que estão sujeitos à compensação: `DELETE CASCATE, DELETE RESTRICT,` e `UPDATE RESTRICT;` DB2 *DataPropagator* não captura atualizações realizadas por qualquer utilitário do banco de dados; DB2 *DataPropagator* não replica dados encripitados e tipos de dados definidos pelo usuário (diferentes dos tipos de dados suportados pelo DB2) deverão ser convertidos antes da replicação.
SQL 2000	Suporte a três tipos de replicação, mas com várias combinações.	Para cada tipo de replicação os agentes possuem comportamentos diferentes (especialmente no processamento *snapshot*) e isso reflete diretamente no cumprimento da restrição de integridade na replicação. Faz uso de *triggers* e *stored procedures* além do protocolo *two-phase commit.*	Desabilita as restrições de chave estrangeira durante a replicação. Desabilita a verificação (*check constraints)* das restrições através de assertivas quando a relação é replicada em um banco de dados remoto.

9.3 Comparação na Distribuição de Dados

Para que haja distribuição de dados, os SGBD's aqui avaliados utilizam componentes que cooperam entre si, integrando os recursos do SGBD para garantir a distribuição de dados, sobretudo no cumprimento das restrições de integridade distribuídas.

A interoperabilidade entre base de dados (heterogênea ou não) é uma preocupação bastante relevante. Os SGBD's avaliados (Ingres, Oracle, DB2 e SQL 2000) suportam componentes da *OpenGroup* (www.opengroup.org), especialmente o *X/Open XA* e o *X/Open DTP* . O *X/Open XA* é um protocolo de interface bidirecional entre o gerente de transações distribuídas e o gerente de recursos do SGBD. Já o *X/Open DTP* é um modelo de arquitetura de software que auxilia no processamento de transações distribuídas. Os fabricantes dos SGBD's aqui comparados são membros da *OpenGroup.* Dessa maneira os SGBD's em questão recorrem a esses recursos de interoperabilidade. Ressalta-se, porém, que somente o uso desses recursos não são suficientes para conduzir a implementação de um ambiente distribuído totalmente confiável. Ao passo em que, as transações globais ainda não podem garantir a integridade dos fragmentos de dados distribuídos.

Um sistema de bancos de dados distribuídos requer um planejamento eficiente independente dos SGBD's envolvidos, que compreenda desde da concepção do projeto até a implementação física dos bancos dados distribuídos (ciclo de vida). Contudo, alguns fatores relevantes são considerados:

- O universo de discurso é o fator determinante para o desempenho do SGBD;
- A transparência de interoperabilidade entre bases heterogêneas é fundamental, pois o convívio com dados legados deve ser considerado;
- A disponibilidade, performance, integridade e segurança dos dados torna o ambiente distribuído viável;
- O custo de acesso aos fragmentos distribuídos e a complexidade de alocação de recursos, são problemas característicos de bancos de dados distribuídos, que ainda não foram solucionados por completo.

- Os mecanismos eficientes para cumprimento das restrições de integridade em relação aos fragmentos distribuídos devem ser abordados;
- Os mecanismos que garantem a segurança lógica e física dos fragmentos são indispensáveis.

Enfim, não seria possível apontar a superioridade de um SGBD em detrimento ao outro, pois encontramos algumas características do contexto distribuído que são melhores implementadas em determinados SGBD's, enquanto que para outros não. Podem ser considerados exemplos dessa constatação quando:

- Determinadas aplicações distribuídas (bancárias, governo, entre outras) exigem características de segurança eficientes, como criptografia e controle de acesso aos dados. Neste caso o Oracle seria o indicado seguido pelo Ingres, SQL 2000 e DB2.
- Aplicações (organizações que não descartam sistemas legados) que possuem integração com bases heterogêneas. Então o SQL 2000 se apresenta como o mais indicado seguido pelo Oracle, Ingres e DB2.
- Aplicações com característica assíncrona de fragmentação de dados (base móveis) ou interação com *mainframes* (sistemas legados sob *mainframe*). Então o DB2 seria mais apropriado seguido pelo Ingres, Oracle e SQL 2000.
- Aplicações que exigem recursos de performance (como *caches* distribuídos) ou utilizam um número elevado de bases distribuídas (empresas com um número elevado de filiais). Neste caso, o Ingres possui mais recursos para esse contexto que o Oracle, SQL 2000 e o DB2.

Portanto, se as aplicações explorarem fortemente os conceitos de fragmentação (horizontal e/ou vertical), surgem dificuldades em implementá-las, sobretudo para as questões de integridade distribuídas, gerenciamento de transações distribuídas e alocações dos fragmentos. Nenhum dos SGDB's analisados possui mecanismos suficientes para assistir aplicações que explorem esses conceitos sem a adição de recursos externos.

CONCLUSÕES FINAIS

A tecnologia de banco de dados e a tecnologia de rede de computadores evoluíram, e estando agora mais acessíveis às organizações. Tal feito, tem alavancado um novo paradigma para banco de dados; e que através de redes de computadores permite às organizações integralizarem suas informações, adequando-as ao seu modelo de negócio.

No início desta dissertação (seção 2.4), que o modelo de bancos de dados distribuídos apresenta vários problemas. Entretanto, eles foram solucionados para o modelo centralizado, mas agora eles estão presentes e mais complexos no ambiente distribuído. Atualmente encontram-se muitas pesquisas na área de banco de dados, mas são poucas as que abordam os problemas do modelo distribuído de banco de dados.

A contribuição deste trabalho consistiu uma busca de elucidação de um dos problemas do modelo distribuído de banco de dados, resultando em uma investigação do controle semântico dos dados distribuídos, envolvendo a definição e o cumprimento das restrições de integridade distribuídas. Mas para isso foi inevitável a verificação das formas pelas quais os SGBD's profissionais (Ingres, Oracle, DB2, SQL 2000) realizam a distribuição de dados (replicação e distribuição), o que resultou na análise comparativa entre eles. Também foi objeto de estudo, o modelo de regras ativas (*eca-rule*) para banco de dados como um mecanismo destinado ao cumprimento das restrições de integridade distribuídas.

Contribuições

Com o entendimento sobre as formas de distribuição de dados suportadas pelos principais SGBD's profissionais, especialmente dos mecanismos de controle de integridade de dados, e ainda com as pesquisas voltadas ao controle semântico de dados distribuídos, esta dissertação destaca as seguintes contribuições:

- Os SGBD's investigados permitem a fragmentação de dados, mas não oferecem recursos eficientes para garantir a semântica dos fragmentos;
- Os SGBD's abordados ainda não conseguem efetuar a verificação das regras de integridade distribuídas sem o auxílio de recursos externos (api's, componentes, etc.);
- A preocupação principal é com o custo de verificação das regras de integridade de maneira global, onde algoritmos são propostos para reduzir esse custo;
- Os serviços de heterogeneidade encontrados nos SGBD's estão bastante avançados contrapondo-se as barreiras proprietárias de alguns fabricantes;
- O modelo de regras ativas é um grande aliado na definição e no cumprimento das restrições de integridade distribuídas;
- Atualmente os projetos de bancos de dados distribuídos devem prever mecanismos adicionais para o cumprimento das restrições de integridade distribuídas;
- O desempenho de um Sistema de Bancos de Dados Distribuídos está diretamente ligado ao projeto de distribuição de dados e alocação de recursos.

Considera-se, entretanto, que a principal contribuição desta dissertação consiste em dizer que os principais SGBD's proporcionam a integração de dados. Mas, que ainda faltam recursos para manter eficientemente essa distribuição sob vários aspectos como: segurança, integridade, performance, custo de comunicação, confiabilidade e disponibilidade. Apesar de tais déficits, é certo que a integração autônoma da informação tem suas aplicações e vantagens.

Trabalhos Futuros

Os argumentos sinalizam que a expansão de pesquisas futuras, voltadas para o contexto de bancos de dados distribuídos, se faz necessária. Percebemos no desenvolvimento desta dissertação, que muitos dos problemas encontrados no ambiente

de bancos de dados distribuídos ainda não foram totalmente solucionados pelos SGBD's, em especial o controle semântico de dados distribuídos. Embora o controle esteja diretamente ligado ao domínio da aplicação, os SGBD's ainda não conseguem garantir eficientemente o cumprimento da integridade distribuída imposta pela aplicação.

Por exemplo, estando fragmentada horizontalmente a relação *Curso*, onde cada fragmento reside em diferentes *sites* (de acordo com as localizações dos campi da universidade), onde o universo de discurso impõe a seguinte situação:

- O curso pode ser oferecido em todos os campi da universidade;
- O valor da mensalidade do mesmo curso é diferenciado em função da região, na qual está inserido o campus da universidade;
- O valor da mensalidade do curso é ajustado conforme a região.

O problema acima não se resolve somente com a manutenção de uma asserção, ou uma *view*, ou ainda com um *trigger* (se o SGBD suportar esses mecanismos com características distribuídas). De qualquer maneira, haverá a necessidade do uso de recursos adicionais para assistir os SGBD's na manutenção e cumprimento da integridade semântica da relação *Curso*.

Neste sentido, é cabível o desenvolvimento de mecanismo genéricos e parametrizáveis (por exemplo, *wrapper*) a qualquer SGBD, com o objetivo de tornar a implementação da integridade semântica distribuída mais amigável, independentemente do domínio da aplicação.

Quanto à análise comparativa da replicação, distribuição de dados e o cumprimento das regras semânticas distribuídas, pode-se definir um ambiente homogêneo (plataforma de hardware e software, configuração, parâmetros e regras de integridade comuns, e massa de dados) para cada SGBD e aplicar um teste comparativo. Essa atividade propende a ser bastante árdua, pois envolverá quatro SGBD's (Ingres, Oracle, DB2 e SQL 2000), todavia os resultados tenderão a serem mais precisos.

Finalizando, há muito a se fazer para que o ambiente distribuído de banco de dados supere as barreiras impostas pelo contexto emergente.

REFERÊNCIAS

[ABB 2000] ABBRY M.ichael; COREY J., Michael; ABRASON Ian **Oracl8i Guia Introdutório** Ed. Campus, 2000, Rio de Janeiro-Brasil, págs. 1 – 41, 2000.

[AGR 1991] AGRAWAL, R.; COCHANE R. e LINDSAY B. "**On Maintaining Priorities in a Producing Rule Language**" na 17a. VLDB 1991. pag. 479 – 487.

[ALO 1994] ALONSO, Gustavo e ABBADI A. El "**Integrating Constraint Management and Concurrency Control in Distributed Databases**". Bulletin of the Tecnical Commitee on Data Engineering, junho 1994, Vol.17 nº.2 IEEE Computer Society.

[ATZ 2000] ATZENI, Paolo, et al.. **Database Systems Concepts, Languages and Architecture.** McGraw Hill Company, 2000. pág. 447 – 460.

[CAI 2000a] COMPUTER Associates Inc. **Ingres/Replication User Guide** download em 13/12/2000. www.cai.com/products.

[CAI 2000b] COMPUTER ASsociates Inc. **Ingres/Star User Guide** download em 13/12/2000. www.cai.com/products.

[CAI 2000c] COMPUTER Associates Inc. **Ingres Database Administrator's Guide** download em 13/12/2000. www.cai.com/products.

[CAI 2000d] COMPUTER Associates Inc. **Ingres Development User Guide** download em 13/12/2000. www.cai.com/products.

[CER 1985] CERI, Stefano e PELEGATTI, Giuseppe. **Distributed Databases Principles and Systems.** International Student Edition MCGraw-Hill. Pag. 173-185. 1985.

[CER 1995] CERI, Stefano; HOUTSMA, A. W. Maurice; KELLER, M. Arthur e SAMARATI, Pierangela "**Achieving Incrmental Consistency among Autonomous Replicated Database**" . Fauve-projeto realizado em Stanford University suportado pelo NSF IRI-9007753.

[CHA 1994] CHAWATHE, S. Sudarshan; MOLINA, G. Hector e WIDOM Jennifer. "**Flexible Constraint Management for Autonomous Distributed Databases**". Bulletin of the Tecnical Commitee on Data Engineering, junho 1994, Vol.17 nº.2 IEEE Computer Society

[CHA 1998] CHAMBERLIN Don **A complete Guide to DB2 Universal Database** IBM Almaden Research Center, Morgan Kaufmann Publishers, Inc. pág. 2 a 90, 1998.

[DAT 1988] DATE C., J. **Banco de Dados – Tópicos Avançados,** 2ª.Ed. Campus, 1988.

[DAT 2000] DATE C., J. **Introdução à Sistemas de Bancos de Dados,** 7ª. edição americana. Pag. 02 a 24 Editora Campus, Rio de Janeiro- Brasil, 2000.

[DEM 2001] DEMPSTER, Robert. **Distributed systems: Distributed Transactions.** Acessado em 20/08/2001 http://saturn.cs.unp.ac.za/~robd/notes/ds/distributed-data/chap14.html

[ELM 2000] ELMASRI, Ramez e NAVATHE, B. Shamkant **Fundamentals of Database Systems.** Third Edition, Addison-wesley.2000 Pág. 4, 74 –100, 766 –793.

[GUN 2001] GUNDERLOY, Mike; JORDEN Joseph L. **Dominando SQL Server 2000 " a Biblia"** págs 2 a 53. Makron Books, São Paulo, 2001.

[GUP 1993] GUPTA, Ashish e WINDM, Jennifer "**Local Verification of Global Integrity Constraints in Distributed Database**". SIGMOD 05/1993, Washington – USA, ACM 1993 pag. 49-58, 1993.

[GUP 1994] GUPTA, Ashish e TIWARI Sanjai, "**Constraint Management On Distributed Design Databases**". NFS IRI-91-16646 , Stanford University, CA 94305, junho 1994, Vol.17 n°. 2 IEEE Computer Society.

[HAN 1993] HANSON, N. Eric e WINDOW Jennifer "**An Overview of Production Rules in Database Systems**" . conferencia internacional de gerenciamento de dados, SIGMOD 1993.

[HEU 1998] HEUSER, C. Alberto **Projetos de Bancos de Dados.** Instituto de Informática da UFRGS, ed. Sagra Luzzatto, pag. 75 a 83. 1998.

[IBM 2000a] INTERNATIONAL Busines MachinE, **Replication Guide References.** DB2 Universal Database, download em 15/02/2001, http://www-4.ibm.com/software/data/db2/library.

[IBM 2000b] INTERNATIONAL Busines Machine, **Users Guide.** DB2 Universal Database, download em 15/02/2001, http://www-4.bm.com/software/data/db2/library.

[IBM 2000c] INTERNATIONAL Busines Machine, **Administration Guide Planing.** DB2 Universal Database, download em 15/02/2001, http://www-4.ibm.com/software/data/db2/library.

[IBM 2000d] INTERNATIONAL Busines Machine, **Distributed Relational Database References.** DB2 Universal Database, download em 15/02/2001, http://www-4.ibm.com/software/data/db2/library.

[KUN 1981] KUN, H.T. RONBINSON, J.T. **On Optimistic Methods of Concurrency Control.** ACM Trans. On Database Systems, vol.8, Novembro de 1981.

[LAM 1990] LAMPORT, L. **Concurrence Reading and Writing of Clocks.** ACM Trans. On Computer systems, vol8, Nov. 1990

[LEW 1998] LEWIS Karry R. e PAPADIMITRIOU H. Christos , **Elements of The Theory of Computation,** 2ª. Edition pág. 292 – 350. Prentice-Hall Inc. New Jersey, USA.

[MAZ 1994] MAZUMDAR, Subhasish e STEMPLE, David "**Helping the Database Designer Maintain Integrity Constraints**". Bulletin of the Tecnical Commitee on Data Engineering, junho 1994, Vol.17 nº.2 IEEE Computer Society.

[MES 1998] MESQUITA, J.S. Eduardo e FINGER Marcelo "**Projeto de Dados em Bancos de Dados Distribuídos**" XIII Simpósio Brasileiro de Banco de Dados – Máringá- Pr. Brasil, pag 87 a 102.1998.

[MIC 2000] MICROSOFT, Press. **Microsoft SQL Server 2000 Administrator's Companion.** "eletronic book" cap. 25 htt://mspress.microsoft.com;

[ORC 2000a] ORACLE Corporation **Oracle8i Replication** release 2 A76959 download em 21/03/2001, http://technet.oracle.com/doc/oracle8i_816/server.816

[ORC 2000b] ORACLE Corporation **Oracle8i Distribution Databases** release 2 A76960 download em 21/03/2001, http://technet.oracle.com/doc/oracle8i_816/server.816

[ORC 2000c] ORACLE Corporation **Aplication Developer's Guide** release 2 download em 21/03/2001, http://technet.oracle.com/doc/oracle8i_816/server.816

[ORC 2000d] ORACLE Corporation **Net8 Administrator's Guide** release 2 download em 21/03/2001, http://technet.oracle.com/doc/oracle8i_816/server.816

[OZS 1996] OZSU M. Tamer e VALDURIEZ Patrick **"Distributed and Parallel Database Systems"** ACM Coputinng Surveys, Vol. 28, nº.1 Março, 1996

[OZS 1999] OZSU M. Tamer e VALDURIEZ Patrick **Principles Distributed Database Systems.** second edition Prentice-Hall, Inc 1999 pág. 1 a 201.

[PAT 1999] PATON W. Norman e DIAZ Oscar **"Active Database Systems"** ACM Coputing Surveys, Vol.31, nº. 1 Março 1999.

[RAM 2000] RAMAKRISHNAN R.; GEHRKE J. **Database Managemnet Systems.** second Edition, 2000 McGraw-Hill International Editions

[REI 2000] REICH, Sigi. **Synchronisation in Distributed Systems.** Distributed Computing and Networks Center – Eletronics e Ciência Computer – University Southamptom - united kingdow http://www.mmrg.ecs.soton.ac.uk pesquisado em 15/12/2000

[SIB 1999] SILBERSCHATZ, Abraham; KORTH F. Henry e SUDARSHAN S. **Sistema de Banco de Dados.** 3a.ed. Makron Books, São Paulo – Brasil 1999 Pág. 191 –208, 589-624, 662-664.

[SQL 2000a] MICROSOFT Corporation **Sql Server Architecture** SQL Server 2000 Books on-line download em 25/04/2001 http://msdn.microsoft.com/library

[SQL 2000b] MICROSOFT Corporation **Conexões** SQL Server 2000 Books on-line download em 25/04/2001 http://msdn.microsoft.com/library

[SQL 2000c] MICROSOFT Corporation **OLE-DB and SQL Server.** SQL Server 2000

Books on-line download em 25/04/2001 http://msdn.microsoft.com/library.

[SQL 2000d] MICROSOFT Corporation **Building SQL Server Applications.** SQL Server 2000 Books on-line download em 25/04/2001 http://msdn.microsoft.com/library.

[SQL 2000e] MICROSOFT Corporation **Transact-SQL Reference.** SQL Server 2000 Books on-line download em 25/04/2001 http://msdn.microsoft.com/library.

[SQL 2000f] MICROSOFT Corporation **Visual Database Tools** SQL Server 2000 Books on-line download em 25/04/2001 http://msdn.microsoft.com/library.

[SQL 2000g] MICROSOFT Corporation **Creating and Maintaining Database.** SQL Server 2000 Books on-line download em 25/04/2001 http://msdn.microsoft.com/library.

[STO 1990] STONEBRAKER, Michael et al. **"On Rules, Procedures, Caching and Views in Database systems"** ACM SIGMOD 1990, 281-290.

[STO 1976] STONEBRAKER, Michael et al. **The Ingres,** University of California, Berkley ACM 1976 – pág. 189

[TAN 1992] TANENBAUM, S. Andrew **Modern Operating Systems.** Prentice Hall, New Jersey, 1994

[WEB 2001a] http://cai.com/products/ingres/analyst/9801/faulkner7473.htm. Faulkner Information Services has granted to Computer Associates International pesquisada em 13/03/2001.

[WEB 2001b] http://msdn.microsoft.com/library/psdk/sql8_ar_ra_6u05.htm **Replication Archicture**. pesquisado em 25/04/2001

[WEB 2001c] http://msdn.microsoft.com/library/psdk/rpltypes_/2f8z.htm
How Snapshot Replication Works. pesquisado em 25/04/2001

[WEB 2001d] http://msdn.microsoft.com/library/psdk/rpltypes_/54j.htm
How Transactional Replication Works. pesquisado em 25/04/2001

[WEB 2001e] http://msdn.microsoft.com/library/psdk/rpltypes_30z/.htm
How Merger Replication Works. pesquisado em 25/04/2001

[WEB 2001e] http://msdn.microsoft.com/library/psdk/rploptions_34ht.htm
Replication Options. pesquisado em 25/04/2001

Apêndice A

Regras Ativas

Os Sistemas Gerenciadores de Banco de Dados tradicionais em suas implementações básicas são passivos e executam as tarefas comuns na base de dados, como: inserção; deleção; consulta; e, atualização, essas tarefas são disparadas por usuários ou aplicações. Entretanto alguns problemas organizacionais baseados em banco de dados não são resolvidos pela passividade do SGBD tradicional, pois a demanda por aplicações mais complexas faz-se necessária ao considerar a relevância das organizações em obterem informações mais precisa e fidedigna de seus negócios. A tecnologia de banco de dados tem evoluído para atender quase sempre a essas necessidades.

Podemos considerar um banco de dados ativo aquele que seu SGBD suporta regras ativas (*triggers*) que normalmente são definidas pela DDL – *Data Definition Language* dando independência ao banco de ados em relação à implementação, ou seja, as modificações comportamentais do banco de dados podem ser efetuadas por uma simples troca de regras ativas, sem a necessidade de modificar as aplicações que as compartilham. Porem quando o SGBD não faz o uso de gatilhos então utiliza se o gerenciamento pelo mecanismo de *polling*. Isto é, um processo de verificação no banco de dados com a finalidade de encontrar se algum evento relevante ocorreu, e, então executar uma nova verificação para as condições e ações necessárias. Esse tipo de mecanismo é muito dispendioso, pois detectar eventos em grandes massas de dados demanda custo de processamento. Em determinados momentos essas verificações não são totalmente confiáveis, pois pode ocorrer o *commit* de uma transação que efetua algum tipo de atualização na base de dados sem que o *polling* tenha sido executado. Outra deficiência do *polling* é que ações do tipo aborto de transação não podem ser implementadas.

O banco de dados ativo suporta e precede aplicações ativas por mover reações comportamentais das aplicações no SGBD, que trata circunstâncias específicas e/ou a

combinação de mecanismos para execução de diferentes funcionalidades. Essas funcionalidades dependem do modelo semântico e também da categoria do domínio de problema. Um banco de dados ativo é modelado para suportar os comportamentos reativos de uma aplicação, o modelo básico é formado por três componentes: Evento, Condição e Ação, também conhecido pelo paradigma *Evento-Condição-Ação* ou *ECA-rule*.

Esta seção conduz um estudo[10] simples e formal baseado no modelo de regras ativas *ECA-rule,* considerando seu funcionamento e aplicações, abordando os seguintes itens: contexto do banco de dados ativo; características estruturais, comportamentais e dimensionais das regras ativas; arquitetura do sistema de regras ativas; regras ativas nos modelos relacional e orientado a objetos; regras ativas distribuídas e desenvolvimento de aplicações ativas.

12.1 Contexto do Banco de Dados Ativo

Contextualizar banco de dados ativo aparentemente é uma tarefa fácil considerando que esse modelo concentra-se apenas nos elementos do *ECA-rule*. Porém a aplicação do paradigma *ECA-rule* envolve situações complexas, as quais surgem em detrimento da demanda pelo uso das regras ativas objetivando soluções de problemas, que quase sempre não são suportadas por modelos passivos de Banco de dados. Nesse contexto as regras ativas atendem às seguintes características de aplicações: aplicações internas do banco de dados; aplicações externas do banco de dados; e aplicações abertas de banco de dados.

- Aplicações internas do banco de dados é um subsistema do SGBD que implementa algumas funções baseadas em regras ativas, neste caso os *triggers* são gerados para sustentar esse subsistema de forma transparente ao usuário. A característica marcante desse mecanismo interno é a possibilidade criar especificações declarativas dessas funções.

10: baseia-se em um amplo trabalho desenvolvido por Norman W. Paton (Universidade de Manchester) e Oscar Díaz (Universidade de Basque Country)

As principais funções incorporadas pelas aplicações de regras ativas internas se destacam: gerenciamento da restrição de integridade; cálculos derivados de dados; controle de replicação de dados; gerenciamento de versões; controle de privacidade, reforço para segurança de dados e gerenciamento de *workflow* [ATZ 00].

- Aplicações externas do banco de dados normalmente expressa o conhecimento específico do domínio de problema, pois as regras são concebidas e predefinidas por esquemas. Também são chamadas de regras de negócio (*busines rules)* onde demonstra a estratégia da organização em atingir seus objetivos. No caso de regras de negócio cada problema é confrontado em separado, ou seja, cada portfólio da organização têm suas regras específicas. Muitas dessas regras às vezes são simples *alerts* ou mensagens para usuários (tomador de decisões) quando uma situação anormal ocorre. Por exemplo, cerca de 50 % dos alunos estão inadimplentes (aplicação instituição de ensino), pedidos não estão sendo entregues na data predeterminada (aplicação de controle de estoque).

- Aplicações abertas de banco de dados, nessa categoria o banco de dados é acionado como um parceiro de sistemas de monitoração (tempo-real). As regras são preparadas para responder às informações oriundas de ambientes externos, *devices* (coletores estratégicos). Neste caso as regras são aplicadas em cenários onde a informação de tempo real é mais relevante que o próprio dado. Fazem uso desse tipo de regras as aplicações com características específicas, como: aplicações médicas, controle de tráfego aéreo, controle de rotas de transportes (exemplo a seguir), controle de linha de produção, dentre outras.

```
ON UPDATE TO ROTA
  IF EXISTS
    (SELECT * FROM Rota WHERE Rota.Placa =
                                Veículo.Placa
         AND(Rota.Posição+Veículo.PosiNew)>
             Rota.ValorFinal)
  DO DISPLAY Veículo.Placa, " Veículo fora da rota"
```

12.2 Características Estruturais, Comportamentais e Dimensionais das Regras Ativas

Nesta seção abordamos o comportamento individual e estrutural de cada componente do modelo das regras ativas *(evento, condição e ação)*, bem como as respectivas dimensões formais do funcionamento. Há também uma preocupação com a descrição do modelo de regras ativas. A representação dessa abordagem é similar a de Paton e Oscar [1999], na qual a notação de apresentação das dimensões é tabular com dois símbolos distintos: o símbolo $\subset$ indica que pode ter mais de um valor apresentado, enquanto que o símbolo $\in$ indica uma lista de alternativas, porém somente um valor deverá aparecer para a dimensão. A tabela 05 expõe a dimensão do modelo de regras ativas.

Tabela 05: Dimensão do Modelo de Regras Ativas

Evento	**Source** $\subset$ { Structure Operation, Behavior Invocation, Transaction, Abstract, Exception, Clock, External } **Granularity** $\subset$ { Member, Subset, Set } **Type** $\subset$ { Primitive, Composite } **Operators** $\subset$ {or, and, seq, closure, times, not} **Consumption mode** $\subset$ { Recent, Chronic, Cumulative, Continous } **Role** $\in$ { Mandatory, Optional, None }
Condição	**Role** $\in$ { Mandatory, Optional, None } **Context** $\subset$ { DB_T, $Bind_E$, DB_E, DB_C }
Ação	**Options** $\subset$ { Structure, Operation, Behavior Invocation, Update Rules, Abort Inform, External, Do Instead } **Context** $\subset$ { DB_T, $Bind_E$, $Bind_C$, DB_E, DB_C, DB_A }

12.2.1 Evento

Um evento é instantaneamente atômico em sua ocorrência, ou seja, ele acontece, ou não acontece, qualquer mensagem disparada por um objeto pode ser um evento em potencial.

Os eventos mais comuns que acontecem no banco de dados são as modificações dos dados que ocorre no próprio banco de dados. Nos sistemas de banco de dados relacionais essas modificações acontecem através dos comandos de *insert, delete, e update*; nos sistemas de banco de dados orientado para objetos surgem através de invocações de métodos [HAN 1993].

O evento é algo que acontece em algum ponto no tempo. Descrever um evento envolve a minúcia de seu acontecimento para que possa se monitora-lo. A natureza da descrição, e o caminho pelo qual o evento possa ser detectado dependem da origem ou do gerador do evento [PAT 1999].

Para dimensão *Source* do evento há um conjunto de alternativas do tipo:

- *Structure Operation*, indica que o evento é causado por uma operação de estrutura do tipo inserção de tupla, atualização de um atributo, ou acesso a um registro.

- *Behavior Invocation*, a execução desse tipo de evento depende de algumas operações definidas pelo o usuário, além disso, o evento pode acontecer antes ou depois que a operação tenha executada. Por exemplo, uma mensagem de *display* pode ser enviada a um objeto qualquer.

- *Transaction*, nesse caso o evento surge através de comandos de transações (*begin, abort e commit)*.

- *Exception,* o evento desse tipo é produzido por algumas exceções. Ex. Permitir o acesso a algum tipo de dado sem a prévia autorização.

- *Clock,* esse tipo de evento ocorre em relação ao tempo, em três situações: programado, (21 de setembro de 2001); relativo (15 dias após o exame dos alunos); e, periódico (quinto dia de cada mês).

- *External,* neste caso o evento depende de um ambiente externo ao banco de dados. Por exemplo, sistema de controle de rota através GPS.

Na dimensão *Granularity* o evento pode ser definido para qualquer tipo de objeto, e em qualquer nível, ou seja, para um conjunto de objetos (*Set*), ou para subconjunto de objetos (*Subset),* ou ainda a alguns membros específicos de um conjunto de objetos (*Members*). A dimensão *Type* é definida pela tipagem dos eventos: *Primitive;* e, *Composite*. Um evento é caracterizado primitivo quando não há nenhum grau de complexidade, normalmente pertence a uma das categorias descrita em *Source*. Por exemplo, o evento

```
ON UPDATE TO Exame.
```

Composite, nesse tipo de evento pode haver combinações de eventos do tipo primitivo, ou ainda o uso de operadores algébricos. Em relação aos operadores pode ocorrer uma variação de sistemas para sistemas. Destaque para os mais utilizados: *disjunction* (E_1 ou E_2); *conjuction* (E_1 e E_2); *sequence* (E_1, E_2); *clousure E, E* é assinalado somente uma vez sem considerar a ocorrência de *E* mais tarde em um intervalo de tampo *Int*; *times* (*n, E*) em *Int,* quando o evento *E* ocorre *n* vezes durante o intervalo de tempo; *not* E_1 em *Int,* detecta a não ocorrência de E_1 no intervalo de tempo *Int* [PAT 1999].

```
ON INSERT TO Exame OR
      UPDATE TO Curso OFF Exame
      IF EXISTS
          (SELECT Curso
             FROM Curso, Exame
             WHERE Exame.Curso NOT = Curso.CodCurso)
      DO ABORT
```

Algumas dimensões cooperam entre si para realizar determinadas atividades. Os eventos do tipo composto além de absorver a dimensão *operators*, também utiliza a dimensão *Consumption mode*, na qual surge a seqüencialização dos eventos. Ao detectar um evento composto (EC) encontramos várias ocorrências de eventos (EV_1 e EV_2), se duas ocorrências do evento EV_1 (primeiro ev_1 e depois ev_1'), tenha sido assinalada e uma ocorrência do evento EV_2 (ev_2). Neste caso as possibilidades de seqüencialização incluem: *sequence* (ev_1, ev_2) ou *sequence* (ev_1', ev_2) ou ainda *sequence* (ev_1, ev_2) $\cup$ *sequence* (ev_1', ev_2). Essas alternativas são tratadas usando a dimensão *Consumption mode*, a qual introduz quatro políticas de consumo:

- *Recent*, nesse contexto considera-se que o mais recente conjunto de eventos primitivos tenha sido usado para construir a composição (EC). (*sequence* (ev_1', ev_2) é detectado quando ev_2 aparece, depois que ev_1 e ev_2 não são considerado para a detecção de EC);

- *chronicle*, essa política consome os eventos primitivos em ordem cronológica (*sequence* (ev_1, ev_2), e é assinalada enquanto ev_2 aparece depois que ev_1 e ev_2 não são considerados para a detecção de EC);

- *Continous*, nesse contexto fecha-se a composição atual e inicia-se uma nova composição com cada evento primitivo acontecendo em seqüência (inicia-se a construção de duas seqüências de eventos enquanto que ev_1 *e* ev_1' aparece, e juntos a seqüência de eventos pode ser assinalada assim que ev_2 é detectado);

- *Cumulative*, nessa política todos os eventos primitivos são acumulados enquanto que o evento composto finalmente aparece (a seqüência do evento é assinalada somente uma vez quando ev_2 aparece, enquanto que o primeiro parâmetro da seqüência é incluído para todas ocorrências de EV_1 (ev_1 e ev_1')) [PAT 1999].

Role, é um tipo de dimensão que indica o papel de detecção do evento para a monitoração, o qual é sugerido pela implementação de diferentes funcionalidades da

ECA-rule. Apenas três alternativas compõem essa dimensão: *optional, none e mandatory.*

12.2.2 Condição

Para os bancos de dados que comportam linguagem de regras, a condição é parte da regra especificada por predicado ou *query* expressa sobre um dado, ou uma coleção de dados do próprio banco de dados.

Quando a condição é uma *query,* e se a *query* produz algum dado, então é natural que a condição foi satisfeita. Isto é implicitamente entendido que a regra é disparada somente quando um novo dado satisfaz a condição[HAN 1993].

No contexto dimensional a condição possui duas situações: *Role e Context* [PAT 1999].

- A dimensão *Role* indica se a condição deve ou não aparecer no *ECA-rule,* portanto a dimensão *Role* pode assumir um dos três valores: *optional, none* ou *mandatory.* No *ECA-rule,* a condição é geralmente *optional.* Quando não há condição para uma *ECA-rule* ou quando o papel é none, então o resultado da regra é *event-action.*

- *Context* indica o ambiente para a condição ser avaliada. Os componentes do *ECA-rule* não são avaliados em separados no banco de dados, pois na execução de uma simples regra esta associada quatro estados distintos do banco de dados (figura 34, pequenos estados): DB_T inicio da transação no banco de dados; DB_E quando acontece o evento no banco de dados; DB_C quando a condição é avaliada no banco de dados e DB_A quando a ação é executado no banco de dados. Com tudo os sistemas de regras ativas podem suportar facilidades na qual a condição da regra pode acessar zero ou mais estados (DB_T, DB_E , DB_C e ainda *bindings* associados ao evento ($Bind_E$).

12.2.3 Ação

O último componente do *ECA-rule*, a Ação, executa as operações específicas no banco de dados quando a regra é disparada e a condição é satisfeita. Ação pode ser construída por uma sofisticada linguagem de *script* ou ainda invocar programas ou métodos.

A dimensão da Ação é composta por dois elementos: *Options;* e, *Context.* A dimensão *Options* específica as tarefas que são realizadas pela ação. As ações podem atualizar alguma estrutura (*structure*) do banco de dados ou um conjunto de regras (*update rules*); executar algum comportamento (*behavior invocation*) contida no banco de dados ou chamadas externa (*external*); informar ao usuário ou administrador do sistema de alguma anormalidade (*inform*); abortar transações (*abort*); utilizar-se de alguma alternativa no curso da ação através *do-instead.*
O *Context* da ação é similar a da condição, a qual indica quais informações estão disponíveis para ação. Pode acontecer da informação ser passada da condição para ação através da DB_E or $Bind_C$ (conforme figura abaixo) [PAT 1999].

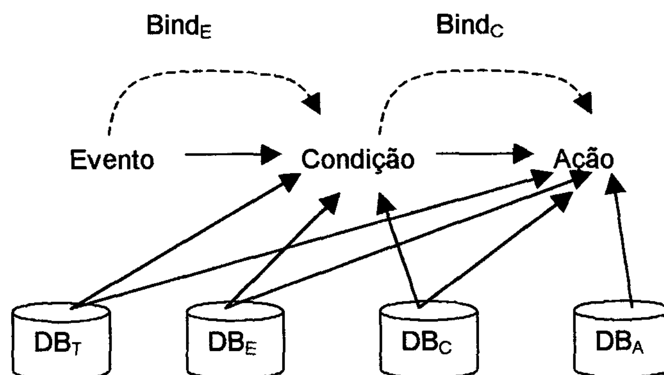

Figura 34: Processamento de Regras em Relação aos Estados do Banco de Dados.

12.2.4 Funcionamento *ECA-rule*

Os passos de execução das regras ativas obedecem a um esquema, todavia as fases de execução do esquema depende do modo de acoplamento entre duas dimensões de execução, *Evento-Condição* e *Condição-Ação.*

A figura 35 apresenta um esquema de elaboração e execução do *ECA-rule*, através da Linguagem de Modelagem UML[11], descartando qualquer influência das linguagens geradoras de regras ativas.

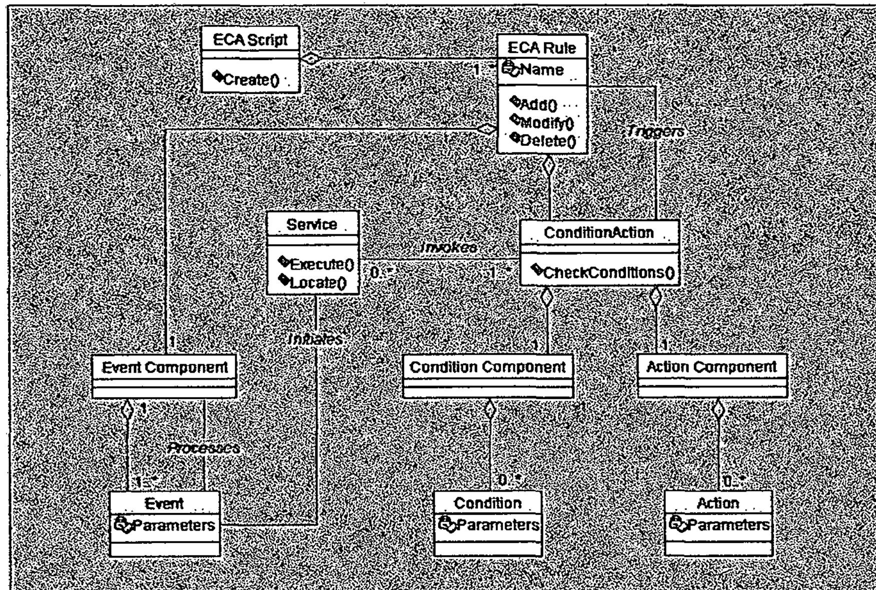

Figura 35: Modelo de Criação e Funcionamento de *ECA-rule*.

A modelo acima apresenta a própria classe *ECA Rule* como a principal classe do modelo (todo). As classes *Event, Condition e Action* estão agregadas aos seus respectivos componentes. Os componentes *Condition Component* e *Action Component* são agregados da classe *ConditionAction,* enquanto que o componente *Event Component* é agregado da classe *Eca Rule* (todo). Por sua vez a classe *ConditionAction* também é agregada à classe *Eca Rule*. Há uma classe de serviço *Service* para suportar a relação entre o evento *Event* e a condição-ação *ConditionAction.* Por fim a classe *ECA Rule* possui uma classe agregada *Eca Script* com a função de criação das regras.

A semântica das linguagens de produção de regras determina como o processamento acontece em tempo de execução dessas regras, porque a interação é arbitraria com o sistema de banco de dados e com as transações que são submetidas por usuários ou programas de aplicações. Considerando também que pode haver um grande volume de regras e que algumas delas podem apresentar um grau elevado de complexidade.

11: UML -*Unified Modeling Language* é uma linguagem para especificação, construção, visualização e documentação de sistemas de *software.* É composta por várias técnicas: FUSION, OMT, OOSE, BOOCH e Statecharts

Nesse contexto a semântica das linguagens de produção de regras é essencial, pois elas apresentam varias alternativas, ressaltando que o domínio de problema é relevante ao uso semântico dessas linguagens.

Um aspecto importante na execução das regras é a resolução de conflito (opção de executar uma regra enquanto que múltiplas regras são disparadas) na ordenação das regras. Em muitos sistemas de banco de dados ativos essa opção é feita arbitrariamente, entretanto para alguns bancos de dados a linguagem de regras providencia características para a regra definindo a resolução de conflito[HAN 93].

A tabela 06 apresenta as dimensões para a execução do *ECA-rule* independente da linguagem de regras suportada pelo SGBD.

Tabela 06: Dimensão do funcionamento do *ECA-rule.*

Condition-Mode $\subset$ { Immediate, Deferred, Detached }
Action-Mode $\subset$ { Immediate, Deferred, Detached }
Transition Granularity $\subset$ { Tuple, Set }
Net-effect policy $\in$ { Yes, No }
Cycle policy $\subset$ { Iterative, Recursive }
Priorities $\in$ { Dynamic, Numerical, Relative, None }
Scheduling $\in$ { All Parallel, All Sequential, Saturation, Some }
Error handling $\subset$ { Abort, Ignore, Backtrack, Contingency }

No modo de acoplamento *Condition-Mode* e *Action-Mode* as respectivas dimensões suportam as seguintes alternativas: *immediate*; *deffered*; e, *detached*. A dimensão *Transition Granularity* tem um papel importante de capturar o relacionamento entre os eventos e as regras. O relacionamento das ocorrências dos eventos em relação à instanciação das regras é de 1:1 ou M:1, contudo a dimensão *Transition Granularity* é assinalada por duas situações: *tuple;* e, *set.*

Uma outra característica que influência no relacionamento entre eventos e disparos de regras é a dimensão *Net-effect policy*, a qual indica se o *net effect* deve ser considerado ou não na ocorrência de eventos. A diferença entre as duas estratégias surge em situações onde varias atualizações do mesmo elemento de dado possa ser considerado como uma simples atualização: se uma tupla é atualizada e então deletada, o *net effect* é de deleção da original tupla; se uma tupla é inserida e então atualizada, o

net effect é de inserção da atualização da tupla; se uma tupla é inserida e logo deletada, o *net effect* não é de modificação para todos[HAN 1993].

A dimensão *Cycle policy* trata da questão do surgimento de eventos assinalados pela avaliação da condição ou ação da regra quando da execução do *ECA-role.* Em geral há duas opções para essa dimensão: *interative;* e, *recursive.*

A dimensão *scheduling* aborda uma questão bastante relevante na avaliação das regras, quando múltiplas regras são disparadas ao mesmo tempo, e, nessa abordagem surgem duas principais situações: *seleção da próxima regra a ser disparada e o número de regras a ser disparada.*

A primeira situação chama a atenção da comunidade de Sistemas Inteligentes, pois é considerada fundamental para entendimento e controle comportamental de um conjunto de regras. Na maioria das vezes a ordenação das regras influência e reflete na espécie de argumentos que acompanham o sistema. Esse contexto menciona as opções da dimensão *Priorities* exemplificando algumas situações de prioridade na execução das regras. A opção *Dynamic* da dimensão *Priorities* apresenta algumas situações, como: prioridade para regras baseadas em recentes atualizações (ex. tempo em que ocorre o evento) ou na complexidade da condição; outra situação é fazer com que o sistema focalize em linha de argumento, o mais recente dado modificado associado ao mais recente disparo da regra. Entretanto, mecanismos disponíveis em sistemas de banco de dados ativos têm se deparado com uma grande quantidade de dados eficientes no contexto, onde o comportamento determinístico é assegurado e muito almejado que tende a suportar esquemas de prioridade, nos quais as regras estão associadas com prioridades estáticas.

Prioridades estáticas são freqüentemente determinadas também pelo sistema (baseada no tempo de criação da regra), ou por usuário assim como um atributo da regra. Nesse segundo caso, a regra é selecionada da coleção simultaneamente das regras disparadas para execução, usando o mecanismo de prioridade (*Priority*). As regras podem aparecer em ordem usando esquema numérico (*numerical*), no qual cada regra tem seu valor absoluto de prioridade, ou por indicação da prioridade relativa (*relative*) das regras declarando explicitamente e determinando que a regra deve ser disparada

antes da outra, enquanto que juntas são disparadas ao mesmo tempo [AGR 1991] [STO 1990].

A segunda situação aborda quatro possíveis opções contidas na dimensão *Scheduling*: *all sequential*; *all paralel*; *saturation*; e, *some*.

A escolha apropriada dentre as quatro opções descrita acima depende do tipo de aplicação em que a regra ou um conjunto delas serão utilizadas. Por exemplo, a primeira opção é bastante confortável para manter a restrição de integridade, pois em qualquer atualização do banco de dados é considerada correta se todas as restrições sejam satisfeitas.

A última dimensão do funcionamento do *ECA-rule Error handling* trata o aspecto de como o tratamento de erro é suportado durante o disparo da regra. Muitos sistemas simplesmente abortam (*abort*) a corrente transação, essa atitude é um comportamento padrão em banco de dados convencional. Alguns sistemas invalidam as regras que depende da existência de um elemento de dado excluído, ou de privilégios revogados. Entretanto há outros procedimentos mais adequados, como: ignore (*ignore*) a regra que apresenta erro, e continua processando outras regras; retroceder (*backtrack*) ao processamento inicial da regra, e recomeçar o processamento da regra ou continuar com a transação; adotar qualquer plano de contingência (*contingency*) na recuperação do estado do banco de dados no caso de erros, possivelmente o uso do mecanismo de exceção do sistema de banco de dados.

12.3 Arquitetura do Sistema de Regras Ativas.

Nessa seção apresentamos uma arquitetura abstrata do sistema de regra ativa, a qual evidência as características descritas nas seções anteriores.

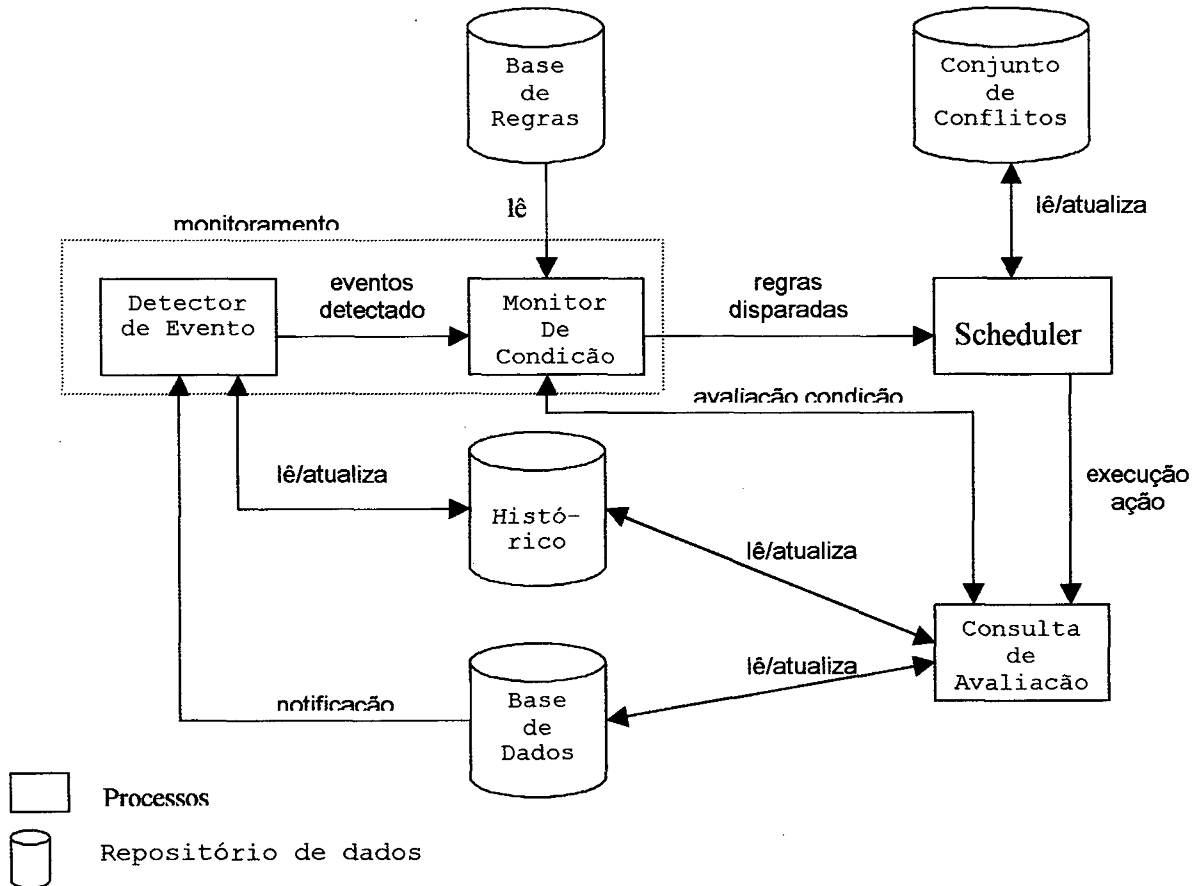

Figura 36: Arquitetura do Sistema de Regras Ativas.

O processo de detecção de eventos verifica se qualquer evento que aparece é de interesse da regra. Eventos primitivos são notificados da base de dados ou de algum ambiente externo; os eventos compostos são construídos a partir dos eventos primitivos mais as informações de eventos anteriores obtidos do histórico. O monitor de condição avalia as condições das regras associadas aos eventos que devem ser detectados pelo detector de eventos. Nos sistemas que suportam regras do tipo *condição-ação,* não há instruções explicitas dos eventos para ser monitorada, embora implementações atuais têm sido feito para monitor de eventos primitivos. O processo de *Scheduler* compara as regras disparadas recentemente com as que previamente serão disparadas, realiza atualizações no conjunto de conflitos, e dispara qualquer regra que estiver programada para ser disparada imediatamente. O processo de consulta de avaliação executa uma consulta ao banco de dados ou ação. O acesso ao banco de dados pode requisitar o estado atual e o estado anterior do banco de dados, e essa ordem é para suportar o monitoramento de como o banco de dados é envolvido [PAT 1999].

A funcionalidade de cada componente da arquitetura depende muito do conhecimento da execução de modelos de banco de dados ativo, o qual será suportado na aplicação a ser desenvolvida. Na arquitetura (figura 36), duas principais categorias de banco de dados ativos são identificadas: *layred* e *integrated.*

Layred, nessa categoria o componente ativo é desenvolvido como uma camada de software imutável sobre um sistema de banco de dados passivo. O benefício dessa categoria é que não obriga o acesso ao código fonte do sistema de banco de dados, e resultando um sistema ativo que pode ser portável para uso em diferentes sistemas de banco de dados passivo. Porém, a falta de interação com o núcleo do banco de dados pode ter impacto direto na *performance* e no limite de funcionalidade, como: detecção de evento primitivo; e, modo de acoplamento e otimização.

Integrated, o componente ativo desenvolvido nessa categoria suporta as mudanças dos sistemas de banco de dados passivos existentes. O desenvolvedor dessa categoria tem mais liberdade em relação ao desenvolvedor da categoria anterior, e esse modelo possibilita o desenvolvimento de banco de dados ativos mais robustos. Embora o modelo possibilita a construção de componentes mutáveis, na prática isso não acontece em grande escala, pois os sistemas são desenvolvidos usando amplamente as camadas de software com "ganchos" para vários núcleos dos sistemas de banco de dados passivo.

12.4 Regras Ativas nos Modelos Relacional e Orientado a Objetos.

Nos sistemas de banco de dados relacionais todos os dados são armazenados em tabelas, e cada tabela contém um número fixo de colunas. As instâncias da tabela ocorrem com a inclusão de tuplas (cada tabela pode ter zero ou mais tuplas). Consultas são disparadas contra o banco de dados através da linguagem declarativa de banco de dados , como *SQL* ou *Quel.* As operações executadas no banco de dados obedecem a um particular predicado recuperando dados através de comparações e junções da várias tabelas. As modificações no banco de dados surgem em detrimento as operações de inserção, deleção e atualização.

O uso de componentes ativos no modelo relacional não é algo novo, há muitos sistemas de banco de dados comerciais (Oracle, DB2, Ingres, SQL Server, e outros) que suportam a implementação de *ECA-rule* através de *triggers* (exceto no Ingres, implementa através do *DBEvent*) . Contudo, há algumas características que modelo relacional ainda não aprimorou em relação a execução de regras, geralmente as regras são disparadas através de operações definidas na estrutura do banco de dados (inserção de tuplas e modificação de tuplas), são poucos os SGBD's relacionais que suportam a execução de uma operação invocada pelo usuário (*Action-Mode* do tipo *deferred*). Por outro lado a extensibilidade das regras ativas para o banco de dados relacional é reconhecida, principalmente para eventos primitivos na execução *stored procedures,* mas a funcionalidade não é suportada pelo modelo discutido (banco de dados ativo).

A linguagem *SQL* tem poder expressivo na construção de condições e atualizações, normalmente as linguagens de construção de regras se configuram como extensão da linguagem de consulta. Em geral os mecanismos propostos para o modelo relacional de banco de dados suporta um ou, uma quantidade limitada do modo de acoplamento, e, um pobre suporte para a detecção de eventos compostos [PAT 1999].

No banco de dados orientado para objetos algumas tarefas de comportamento ativo que são realizadas com dificuldades no modelo relacional, agora suportadas pelos métodos dos sistemas orientados a objetos, outra característica do modelo orientado a objetos que pode favorecer a implementação de regras ativas é o encapsulamento escondendo a estrutura do *ECA-rule.* Por outro lado, são poucas e prematuras as pesquisas realizadas na extensão do banco de dados orientado a objetos (banco de dados orientado a objetos ativo), isso deve ao fato, de que, o modelo relacional possui a maior escala em uso, enquanto que o uso do banco de dados orientado a objeto, na maioria das vezes destina-se à domínios de problemas não tradicionais.

Os bancos de dados orientados a objetos possuem mais poder do que a maioria das extensões dos modelos relacionais, a diferença mais comum é que eventos primitivos em banco de dados orientados a objetos ativos são muitas vezes associados com a invocação dos métodos, especialmente para aqueles acessos que atualizam estruturas de objetos. Isso é um desejo de se prevenir redundância e independência de dados, pois o

comportamento ativo é ligado diretamente na estrutura dos objetos, e de
certa forma o fato de que alguns sistemas usam arquitetura de camadas, ou seja, construir eventos dentro das bases de estrutura de operações, isso não é recomendável [PAT 1999].

Há várias pesquisas realizadas por [HAN 1993] e [PAT 1999] com mais profundidade abordando vários sistemas relacionais como: *Starburst,* POSTGRES, *Ariel,* SQL-3, RPL, e também em projetos de banco de dados orientado a objetos ativo, destacando: HiPAC, EXACT, NAOS, Chimera (não são baseado na extensão de C++[12]), Ode, SAMOS, Sentinel e REACH (são baseados na extensão de C++). Essas pesquisas não serão tratadas nesse escopo.

12.5 Regras Ativas Distribuídas.

Até o momento essa abordagem concentrou-se em mostrar algumas características e propriedades do *ECA-rule* em ambientes centralizados. Mas, novas e modernas aplicações têm surgidas com aspecto distribuído, por isso a proposta dessa seção é explorar algumas situações do comportamento ativo em sistemas distribuídos, referenciado pelo modelo padrão de comunicação entre objetos através de autônomos nodos de rede computadores, CORBA[13].

Detecção e gerenciamento de eventos, ou gerenciamento de regras são vistos como serviços cooperativos oferecendo a funcionalidade de empacotamento como um monolítico mecanismo ativo[PAT 1999].

A figura 37 apresenta a arquitetura do *ECA-rule* para um ambiente distribuído, a qual demonstra que o sistema suporta detecção de bases do *ECA-rule,* processamento, identificação de eventos e complexas situações, que são realizadas por sistemas suportados pelo modelo CORBA combinando informações heterogêneas oriundas de diversos locais.

12: Derivada da linguagem de programação C, C++ é uma linguagem híbrida, que implementa os conceitos de orientação a objetos tanto quando o modelo estruturado.
13: CORBA – Common Object Request Broker Architecture, é o um padrão da OMG – Object ManegementGroup, propõe total homogeneidade entre os aplicativos que suportam esse modelo.

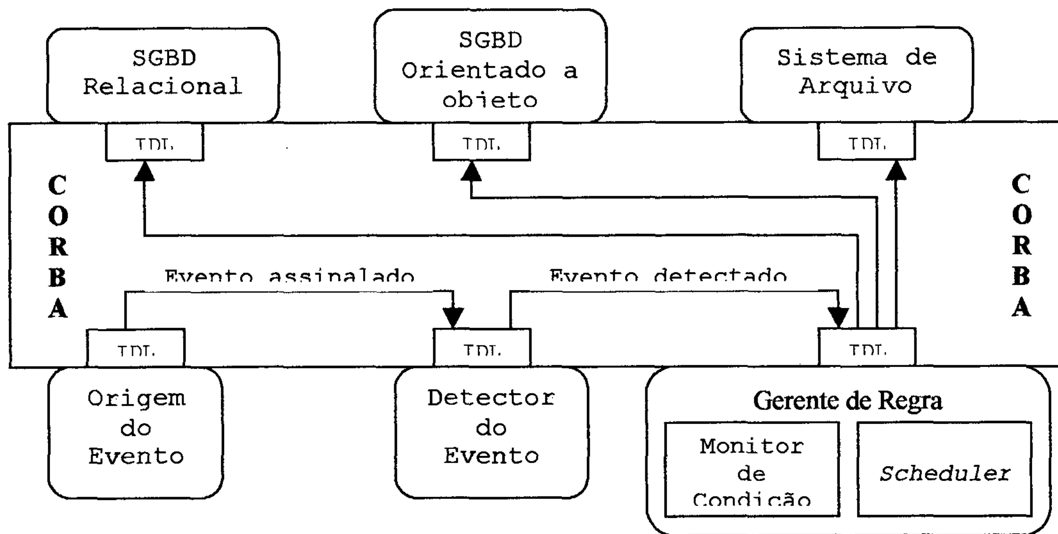

Figura 37: Arquitetura do *ECA-rule* Distribuída.

A figura acima sugestiona que cada componente seja centralizado e localizado por diferentes *sites*, no entanto não há necessidade de ser dessa forma. Para que a detecção da instância de eventos possa ser distribuída entre os *sites* é necessário que cada *site* contenha um detector de evento local e um detector de evento global. A idéia básica corresponde a dois estágios, primeiro o detector encontra o evento no sistema centralizado e depois se responsabiliza em monitorar os eventos compostos entre *sites*. Um detector global de evento responsável pelo evento *E* registra seus interesses em *E's* componentes corresponde detector. A detecção de um evento global é distribuída em diferentes *sites*. Logo que o evento é detectado, um sinal é enviado para cada detector que registrou o interesse no evento. O gerenciamento de regras é suportado pelo mesmo contexto e oferecem alternativas diferentes: uma gerência de regras mantém a base de regras global; um conjunto de regras gerenciadas em *sites* distintos cada um com base local; e, conjunto de gerência de regras com um conjunto global.

No contexto distribuído para regras ativas devemos considerar o *overhead* para as alternativas estruturais, mesmo no CORBA que oferece serviço de eventos e suporta comunicação *asynchronous* entre objetos. A comunicação acontece através do canal de eventos (*event channel*). O canal de eventos disponibiliza em fila os eventos que são fornecidos pelo fornecedor de eventos, e remove os eventos consumidos pelo consumidor de eventos. O funcionamento do canal de comunicação utiliza dois modelos

para iniciar a comunicação. O modelo *push,* que permite ao fornecedor de eventos iniciar a transferência dos dados do evento para o consumidor, já o modelo *pull,* permite ao consumidor de eventos requisitar os dados do evento, do fornecedor de evento. Isso caracteriza que os responsáveis explicitamente pela inicialização da execução de eventos são os fornecedores e consumidores de eventos, além disso, o padrão CORBA não suporta explicitamente eventos compostos. Contudo esse mecanismo de comunicação deveria endereçar a comunicação entre diferentes componentes suportando a funcionalidade ativa. Para as instâncias de eventos deveria sinalizar a fase envolvendo a comunicação entre a origem do evento, e o detector de eventos seguindo o modelo *push* (assim como as reativas), ou o modelo *pull* (assim como as pro ativas). Da mesma forma a comunicação entre o detector de eventos e a gerência de regras também segue a uma abordagem pro ativa ou reativa. A forma reativa implica a subscrição explicitamente do gerente de regras para um conjunto de eventos dos quais ele está interessado, enquanto que a abordagem pro ativa libera o gerente de regras para podar a coleção de eventos periodicamente.

Suportar o modelo distribuído de regras requer eficiência, pois o conhecimento do ambiente é fundamental, a heterogeneidade da origem dos eventos, a diversidade das origens de dados, e, a falta de um estado simples. Com essas características a execução de regras ativas torna uma tarefa desafiadora, onde requer precisão e sincronismo.

Nos sistemas centralizados um simples *clock* permite a ordenação das instâncias dos eventos baseada em cada ocorrência de tempo. No sistema distribuído não se tem um tempo global, cada *site* possui seu próprio *clock* e os eventos podem ocorrer simultaneamente em diferentes *sites.* Eventos de diferentes *sites* podem compor um evento do tipo composto, e então deveria ter eventos de *timestamp* globalmente significante. Entretanto *clocks* locais são sincronizados através do tempo especial do servidor que transmite a informação de tempo para cada *site,* o atraso de sincronização deveria ser inferior a um parâmetro *P,* obtendo o máximo da diferença de tempo entre os dois *ticks* de qualquer um dos dois *clocks* locais. A soma de dois *timestamp* são requisitados e chamados de *local timestamp,* necessários para construção de eventos compostos no *intrasite,* e o *timestamp* global é usado na forma canônica para detectar eventos compostos entre *sites* [SCH 1996].

Outra questão é o tratamento de erros, a execução do modelo deve encarar a interrupção e o atraso na comunicação de qualquer mensagem entre os componentes do mecanismo ativo. A solução depende muito dos componentes e do impacto do erro. Para a espera da comunicação durante a sinalização, entre o detector de evento e a origem do evento, pode ocorrer que o detector de evento global comesse receber eventos em qualquer ordem da atual ocorrência. Podendo optar por contingência de ações do tipo avaliação *assynchronous* ou avaliação *synchronous* [SCH1996].

12.6 Desenvolvimento de Aplicações Ativas.

Essa seção aborda algumas dificuldades no desenvolvimento de aplicações, bem como uma breve descrição do projeto, análise e implementação de regras ativas. As dificuldades encontradas, são:

- Não é muito claro a identificação da(s) parte(s) da aplicação que suportará o uso de mecanismos ativos, e também quais outras técnicas que possam ser usadas comitantemente, e ainda ter noção se o uso das regras podem influenciar na performance do sistema. Estas questões surgem em detrimento de não haver uma metodologia apropriada para projetos de regras ativas;

- Dificuldade de entendimento da funcionalidade de uma considerável base de regras, pouca descrição do fluxo de controle das regras, muitas vezes as regras se interagem em caminhos complexos, e não é simples descrever o fluxo de controle através da uma aplicação;

- A baixa expressividade de ferramentas associadas às regras ativas, pouco suporte de monitoramento, navegador ou *debugador* de regras ativas.

Essas questões reafirmam a necessidade de metodologias de projetos e técnicas de análise para a produção de *ECA-rule,* assim como ferramentas de suporte. Contudo algumas explanações sobre projeto, análise e implementação de regras ativas serão traçadas a seguir.

12.6.1 Projeto de Regras Ativas

Em projetos de banco de dados encontramos modelos, que descrevem aspectos estruturais do domínio da informação para o banco de dados (ER e ER Estendido). Na descrição de processos há varias ferramentas consolidadas pelos projetistas (Diagrama de Fluxos de Dados e Diagrama de Classe) para abstrair situações do mundo real. No entanto, os comportamentos ativos muitas vezes estão diretamente ligados as estruturas armazenadas no banco de dados, e as ações das regras são encaradas como processos ou execução de uma tarefa. E mesmo assim até o momento não há técnicas ou adaptações menos complexa para algumas técnicas já utilizadas no suporte às funcionalidades das regras ativas; mesmo considerando a particularidade de cada aplicação. Mas várias proposta e métodos estão sendo feitos para suportar explicitamente o comportamento ativo, mas eles muitas vezes são extensões de modelos (ER^2) já existente (Navathe, et al. 1995) que suporta a descrição de eventos e regras que são mapeados para as regras ativas, similar encontradas em banco de dados comerciais. Uma outra proposta é IFO2 (Abitebul e Hull 1987) uma adaptação do modelo de construção IFO, para o uso de descrição de eventos compostos e também inclui mapeamento para linguagem de regras ativa. Um outro caso, é extensão da modelagem de linguagem orientada a objetos, proposta por Bichler e Schrefl (1994) para suportar eventos temporais.

Embora todas essas abordagens assumem as regras ativas, no entanto o método de projeto deixa em aberto, quais as partes da aplicação deverá usar as técnicas de regras ativas, e quando usar outras alternativas. Essa foi a preocupação do IDEA, metodologia de projetos (Ceri e Fraternelli, 1997). O qual sugere os *insights* iniciais apara aplicação, e providência uma descrição de alto nível para a linguagem, no segundo estágio do desenvolvimento são mapeados para baixo nível dos *triggers*.

12.6.2 Análise do *ECA-rule*

Se em projetos do *ECA-rule* encontramos dificuldades, na análise a dificuldade é ainda maior, porque algumas propostas são incompletas ou ambíguas para suportar a semântica do banco de dados ativo, isso devido ao fato de que, algumas literaturas são escritas informalmente, é caso de Stronebraker et al. (1990), e Widom e Finkelstein

(1990). Devido à informalidade encontrada na análise de regras, recentes especificações formais têm se desenvolvido para sistemas de banco de dados ativo usando denotação semântica (Widom 1992; Coupay e Collet 1995), Object-Z (Campin et al. 1997), e combinando técnica de lógica/operacional (Fraternali e Tanca 1995; Fernandes et al. 1997), mesmo esclarecendo a informalidade na especificação de sistemas ativos, elas ainda não estão sendo usadas como argumentação sobre as bases de regras. No entanto há algumas características encontradas no comportamento de regras, como: *Termination; Confluence; e, Observable deteminism*.

Termination é uma característica de análise de regra que, através de um gráfico cíclico tenta identificar quais as regras podem falhar para terminar. Esse ciclo é apontado no grafo (figura 38) onde os nodos representam as regras e as arestas representam os relacionamentos *can-trigger*.

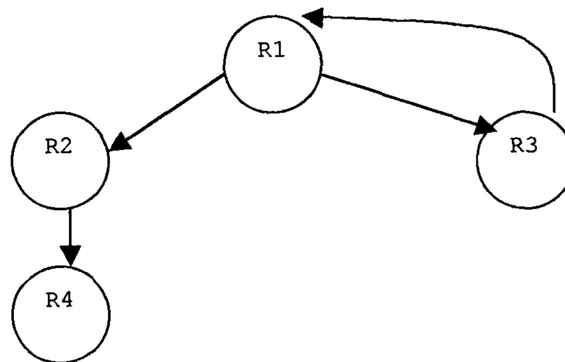

Figura 38: Grafo de Dependência de Disparo de Regras.

Na figura acima é evidente que qualquer disparo das regras, R2 e R4 terminará normalmente, mas o disparo das regras R1 e R3 pode iniciar uma série de disparo de regras que para terminar terão que falhar.

Na *Confluence* o resultado do processamento é analisado independente da ordem em que simultâneas regras foram disparadas, e selecionadas para o processamento. Uma forma de entender a confluência é considerar que o disparo de uma regra em um estado do banco de dados pode levar o banco de dados a novos estados. Se mais de uma regra pode ser disparada em qualquer tempo, então sucessores estados do banco de dados pode existir. A figura abaixo ilustra esse caso onde os nodos são estados *S* e arestas são disparos das regras *R*.

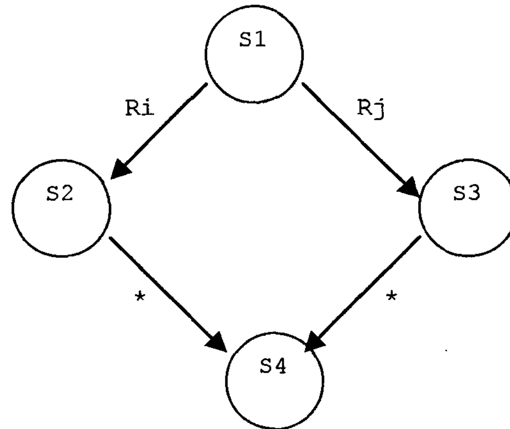

Figura 39: Grafo de Confluência do Comportamento de Regras.

Na figura 39 os estados S2 e S3 são sucessores de S1 resultado do disparo das regras Ri e Rj respectivamente. A base de regras é confluente se, para qualquer das duas regras Ri e Rj disparadas em qualquer estado inicial de S1, um simples estado final S4 é garantido sem considerar a ordem subseqüente, e simultaneamente as regras são engatilhadas e selecionadas para disparo (* nas arestas figura 39). Uma outra consideração possível, é que as ações de Ri e Rj pode ter disparado outras regras na transição do estado S1 para os estado S2 e S3, e que o comportamento dos subseqüentes disparos dessas regras são computadas como entrada na base de regras em detrimento da ocorrência da confluência [PAT 99].

O papel do *Observable determinism* é observar o efeito do processamento das regras por um usuário, ou um sistema (analisador de regras), independente da ordem em que as regras são engatilhadas e selecionadas para processar. Essa questão procura entender o comportamento determinístico dos sistemas de regras que surgem das fronteiras do próprio banco de dados. No exemplo a seguir duas regras implementam a reação de confluência, mas não é determinístico.

```
ON <evento E1>
  IF <condição C1>
    DO <envia mensagem para o usuário>
  ON <evento E2>
    IF <Condição C2>
      DO <ABORT>
```

Se a primeira regra é marcada para disparar antes da segunda, então o usuário recebe a mensagem e a transação no banco de dados é abortada. Se a segunda regra é sinalizada antes da primeira, então a transação é abortada, mas a mensagem não é enviada ao usuário.

12.6.3 Implementação

Muitos usuários podem relutar em aplicar as facilidades ativas, porque antecipados problemas surgem com a manutenção, comportamento imprevisíveis, falta de controle, dentre outros. Cada aplicação pode ter uma ou várias linguagens para implementação da *ECA-rule,* e em diferentes níveis. Por exemplo, *O HIPAC,* foi implementado em três diferentes linguagens, uma utilizando a linguagem *C,* outra em *Smalltalk-80* e também em *Lisp*, e todas elas são integradas ao SGBD convencional. Outra linguagem bastante explorada e o C++, principalmente na implementação de banco de dados orientado a objetos ativo. Linguagens de outros segmentos (IA- inteligência artificial) também são aproveitadas para a produção de *ECA-rule*, é o caso do *OPS5, PRO*LOG e *Lisp*. Além das linguagens, há também o uso de algoritmos, que implementam mecanismos específicos nos sistemas ativos, é o caso do *A-TREAT* usado na execução de testes da condição no *Ariel.*

Com a variedade de linguagens imperativas, possibilidade do uso de algoritmos em situações específicas na implementação, e, considerando que quase sempre há necessidade de integração com a linguagem declarativa do banco de dados. Logo, muitos desenvolvedores defendem a importância do uso de *debugadores* na implementação do *ECA-rules.* O problema é que os modelos de debugadores tradicionais não são adequados para *debugar* regras. Porque as linguagens de programação tradicionais possuem um controle seqüencial e é especificado explicitamente e estaticamente pelo programador, enquanto que as regras ativas são disparadas dinamicamente pelo sistema baseado no fluxo de eventos [PAT 99].

Mesmo com a dificuldade de se construir um debugador que satisfaça as características de um mecanismo ativo, algumas ferramentas para essa finalidade foram apontadas. É caso do DEAR, um *debugador* do sistema EXACT (Diaz et al. 1994) que

tenta abordar a questão mesclando ciclo de regras e eventos. Nesse caso o usuário do *debugador* deve certificar não só as regras que foram disparadas, mas também quais eventos disparam as regras que aparecem na instrução da transação, ou na execução da própria regra.

Outra ferramenta surge no sistema Sentinel (Chakravarthy et al. 1995), a ferramenta de *debugar* (pós-execução) simula uma execução ativa consultando arquivos de *log*, armazenados quando da execução do mesmo. Essa ferramenta focaliza e mostra a intercalação entre eventos e regras, mas ainda é restrito a eventos primitivos. A monitoração de eventos compostos é uma tarefa mais complexa, um grande número de ocorrências pode ser necessário para mostrar os confusos caminhos, nos quais os eventos primitivos são combinados para obter a ocorrência dos eventos compostos.

Conclusão

Embora o modelo *ECA-rule* tenha inúmeras aplicabilidades, ele pode ser um forte aliado aos SGBD's para assistir a definição e o cumprimento das regras ativas, tanto para o ambiente centralizado quanto para o distribuído.