JULIANA PASQUAL

USO DE XML PARA INTEROPERABILIDADE ENTRE BASES HETEROGÊNEAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre, Curso de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.

Orientador: Marcos Sfair Sunye

CURITIBA

2002

Ministério da Educação
Universidade Federal do Paraná
Mestrado em Informática

PARECER

Nós, abaixo assinados, membros da Banca Examinadora da defesa de Dissertação de Mestrado em Informática, da aluna *Juliana Pasqual*, avaliamos o trabalho intitulado, *"Uso de XML para Interoperabilidade entre Bases Heterogêneas"*, cuja defesa foi realizada no dia 28 de agosto de 2002, às quatorze horas, no anfiteatro A do Setor de Ciências Exatas da Universidade Federal do Paraná. Após a avaliação, decidimos pela aprovação da candidata.

Curitiba, 28 de agosto de 2002.

Prof. Dr. Marcos Sfair Sunyé
DINF/UFPR - Orientador

Profª. Dra. Maria Salete Marcon Gomes Vaz
UEPG - Membro Externo

Profª. Dra. Laura Sanchez Garcia
DINF/UFPR

MINISTÉRIO DA EDUCAÇÃO - UNIVERSIDADE FEDERAL DO PARANÁ - Setor de Ciências Exatas - Coordenação de Pós-Graduação em Informática

AGRADECIMENTOS

Ao Prof. **Marcos S. Sunye**

Por sua orientação e confiança que nunca falharam.

Aos meus **pais**

Pelo incentivo e motivação que me ajudaram a terminar este trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E LISTA DE SIGLAS

ERC+	–	(Modelo) Entidade Relacionamento Complexo.
DOM	–	Domínio de um conjunto de valores (idéia matemática).
DTD	–	Document Type Definitions.
SGBD	–	Sistema de Gerenciamento de Banco de dados.
XML	–	Extensible Markup Language.
RWS	–	Real World Stats.
SQL	–	Structured Query Language.
VIQUEN	–	(Protótipo) Visual Query Enviroment.
VQL	–	Visual Query Language.
W3C	–	World Wide Web Consortiun.

RESUMO

A coexistência de bancos de dados heterogêneos, implementados sobre diferentes modelos de dados, com linguagens, representações e plataformas diversas, inevitável em muitas empresas e organizações, tornou natural o uso de bancos de dados integrados para compartilhamento de informações, tanto na tomada de decisão quanto na manipulação de dados. Com o objetivo de proporcionar interoperabilidade entre o banco de dados integrado e os bancos de dados locais que o compõem, de forma transparente e mantendo a autonomia destes, sem que seja necessário uma reestruturação, este trabalho apresenta uma solução baseada em XML. O resultado deste trabalho é a decomposição de uma expressão SQL, apresentada ao esquema integrado por intermédio de um sistema de visualização de esquemas, em expressões SQL correspondentes a cada esquema local. A solução apresentada compreende: 1) o mapeamento do esquema integrado em relação aos seus esquemas componentes, este mapeamento utiliza a estrutura da linguagem XML, devido a seu poder e a sua flexibilidade semântica e por ela possuir um padrão para troca de informações independente de plataforma ou tecnologia; 2) um algoritmo baseado nas regras de integração, utilizadas para gerar o esquema integrado, e nas informações de cada esquema local, obtidas no mapeamento; 3) a utilização de uma estrutura de modelo tanto relacional quanto objeto/relacional, abrangendo os conceitos de classe, herança, atributos multivalorados e compostos.

ABSTRACT

The coexistence of heterogeneous databases, implemented using different data models, with diversity of language, representation and hardware platforms, become natural to use the technology of database integration for sharing information. With the purpose to provide interoperability between the integrated database and the local databases that compose it, preserving the autonomy of local databases, this work presents a solution based on XML. The result of this work is a decomposition of a SQL expression, introduced to integrate schema, into SQL expressions correspondents to each local schema. The solution presented proposes: 1) the mapping of integrated schema in relation of its components schemas, using the structure of XML language, because of its power and its semantic flexibility and because the XML has a standard for changing information independent of platform or technology; 2) an algorithm based on integration's rules, used for create the integrated schema, and based on information about each local schema of mapping; 3) the utilization of both relational and object/relational model structure and the concepts of class, inheritance, multivalued and composed attribute.

1 INTRODUÇÃO

Com o crescimento das organizações e o aumento de competitividade existente no mercado, muitas companhias inicialmente centralizadas, fragmentaram-se em unidades menores descentralizando a gerência dos negócios e, consequentemente, descentralizando seus dados e processamentos. Assim, cada unidade passou a possuir e controlar seus próprios bancos de dados.

Diferentes culturas organizacionais, fusões de empresas ou até mesmo falta de planejamento também contribuem para o surgimento de diversos bancos de dados em uma organização. Normalmente, estes bancos de dados são heterogêneos e implementados utilizando diferentes modelos de dados (como relacional, hierárquico ou orientado a objeto), com linguagens, representações, tecnologias e plataformas diversas, que muitas vezes modelam objetos idênticos ou similares.

Nos dias de hoje, a coexistência destes diferentes bancos de dados em uma organização tornou-se natural e inevitável. Porém, surgiu a necessidade de compartilhamento destas informações, tanto para tomada de decisões, quanto para manutenção dos dados.

Diante disso dá-se a importância de desenvolver mecanismos que permitam interoperabilidade entre estes bancos de dados, permitindo ao usuário não somente consultar diversos bancos de dados de forma transparente, como também manipular seus dados, podendo alterá-los, excluí-los e até mesmo incluir novos dados. Estes mecanismos devem também preservar a autonomia local dos bancos de dados, mantendo sua integridade e coerência.

Com o objetivo de suprir a necessidade de interoperabilidade entre bancos de dados heterogêneos, muitas linhas de pesquisas se destacaram (Spaccapietra; Parent, 1994). Uma delas propõe um esquema global, onde esquemas correspondentes a cada banco de dados local representam uma visão coerente destes e são combinados em um único esquema integrado. Estes esquemas devem ser

representados em um modelo de dados comum de modo a facilitar a especificação do esquema integrado. Devem também denotar semanticamente os bancos de dados que serão integrados. O retorno deste processo é um esquema integrado representando os bancos de dados subjacentes, com cada um de seus objetos devidamente relacionados, via mapeamento, aos objetos correspondentes nos bancos de dados locais. Através deste esquema integrado e seu mapeamento será possível a manipulação de dados de forma transparente, ou seja, o usuário não precisará estar ciente da existência de vários bancos de dados, nem da localização dos objetos nestes.

O processo de integração se dá através de fases, a primeira delas é a pré integração, onde cada esquema individual é traduzido em um esquema utilizando um modelo comum. A segunda fase é identificar os objetos relacionados ou em conflito e classificar suas relações. Após identificados e classificados os objetos e os esquemas traduzidos para um modelo comum, uma metodologia de análise é aplicada. O modelo integrado é gerado a partir de regras de integração, entre elas, Disjunção, Inclusão, Interseção e Equivalência.

Existem inúmeros trabalhos e metodologias propostos para a problemática de integração de bancos de dados heterogêneos (ver Capítulo 2). Porém, poucos pesquisadores se dedicaram ao processo de manipulação dos dados a partir do esquema integrado. Além da dificuldade de mapeamento entre o esquema integrado e os esquemas componentes, esbarra-se também no problema de controle de concorrência e transações entre os diversos bancos de dados envolvidos. Mesmo ferramentas comerciais existentes hoje no mercado, permitem alteração de informações somente quando trata-se de uma regra de equivalência e não controlam transações quando trata-se de plataformas diferentes.

Para que a autenticidade e a coerência nos diversos bancos de dados seja mantida é necessário um controle de transação e concorrência de forma global. Uma transação global é composta por um conjunto de subtransações que assegura a viabilidade dos dados e é responsável pela correta atualização dos dados em todos os

bancos de dados locais. O controle de concorrência global é um componente que coordena a execução das subtransações de forma global para que a consistência dos dados seja mantida. Estes segmentos não violam a autoridade local dos bancos de dados envolvidos, isto é, os bancos de dados locais não perdem o controle sobre seus dados e principalmente não precisam ser reestruturados para permitir interoperabilidade através de esquemas integrados.

A proposta deste trabalho é viabilizar a manipulação de dados através de expressões globais no esquema integrado. Estas expressões são traduzidas para sub-expressões de forma individual para serem submetidas em cada banco de dados subjacente.

A estrutura do trabalho permite que o modelo do esquema integrado seja tanto o relacional quanto o objeto/relacional, o qual adiciona ao paradigma do modelo relacional conceitos de objeto complexo (multivalorados e compostos), além dos conceitos de classe e herança.

Será apresentado um algoritmo baseado nas regras de integração, utilizadas para gerar o esquema integrado, e nas informações obtidas de cada banco de dados local. Também será proposto um mapeamento entre o esquema integrado e os esquemas locais envolvidos. Ambos abordando os conceitos do modelo de dados objeto/relacional.

Como dito anteriormente, para que seja possível a tradução de expressões entre o esquema integrado e os diversos esquemas locais, é necessário que haja um mapeamento entre eles. Neste trabalho, o mapeamento será expresso através da linguagem de marcação eXtensible Markup Language[1], XML (MARCHAL, 2000), por ser uma linguagem com grande flexibilidade e poder semântico e possuir um conjunto de padrões para troca de informações de forma estruturada, independente de plataforma ou tecnologia.

---

[1] Linguagem de Marcação Estendida

As expressões serão escritas na linguagem SQL, Structured Query Language[2], utilizada pela grande maioria dos Sistemas de Gerenciamento de Banco de Dados, SGBDs, atuais. E também devido ao fato da nova especificação da SQL (ISO/IEC 9075, 1999) possuir recursos que abrangem as principais características do modelo objeto/relacional, como abordado na Capítulo 3 deste trabalho.

O trabalho segue, como padrão de entrada para as expressões, um sistema de visualização de esquemas – o VIQUEN[3] (GUEIBER, 2001). O VIQUEN é uma aplicação de consulta visual que opera sobre esquemas relacionais e que utiliza o modelo objeto/relacional para interagir com o usuário. O VIQUEN permite somente consultas a uma única base de dados, através do mapeamento entre os operadores algébricos do modelo objeto/relacional para o SQL.

Esta abordagem mostra-se vantajosa por adotar um padrão genérico, tanto para a linguagem das expressões quanto para o mapeamento, compreendido pela maioria dos SGBDs atuais.

1.1 CONTRIBUIÇÃO DO TRABALHO

Com este trabalho pretende-se apresentar uma maneira de alterar dados em bancos de dados heterogêneos, através de um esquema integrado, utilizando expressões em SQL. Estando este esquema integrado no modelo objeto/relacional e o mapeamento entre o esquema integrado e os bancos de dados heterogêneos locais sendo feito através da linguagem de marcação XML e suas facilidades.

Esta dissertação esta organizada em mais sete capítulos. O Capítulo 2 apresenta alguns trabalhos correlacionados, que tratam de interoperabilidade entre banco de dados e a contribuição pretendida em relação a estes trabalhos. O Capítulo 3

---

[2] Linguagem de Consulta Estruturada

[3] VIQUEM – do inglês Visual Query Enviroment – Ferramenta de Consulta Visual.

discorre sobre o formato padrão da linguagem SQL. O Capítulo 4 descreve os principais pontos que devem ser analisados no momento da integração de bancos de dados. Apresentando, também, um exemplo da implementação de uma integração de bancos de dados heterogêneos. O Capítulo 5 descreve o controle de transação em um sistema de banco de dados integrado. O Capítulo 6 apresenta a linguagem de marcação XML, sua estrutura e principais características. O Capítulo 7 destaca o mapeamento e o algoritmo que formam a estrutura básica desta dissertação, ou seja, que propõem uma solução para a interoperabilidade entre banco de dados através da decomposição de expressões. E por fim, o Capítulo 8 descreve os resultados obtidos com este trabalho bem como os trabalhos futuros.

2 ESTADO DA ARTE

Vários projetos para sistemas de bancos de dados integrados têm sido desenvolvidos através dos anos, tanto ferramentas comerciais quanto protótipos acadêmicos. Tais sistemas são limitados pela funcionalidade ou pela especificidade das ferramentas.

Produtos como SQLServer, Oracle e Ingres/Star são exemplos de ferramentas comerciais. O SQLServer proporciona duas linguagens para múltiplos bancos de dados: Transac-SQL e Visual Query Language (VQL). Através destas linguagens é possível qualificar o nome de um relacionamento com o nome de um banco de dados ou definir visões, especificar procedimentos ("*queries*") entre os vários bancos de dados e manipular dependências. Oracle possui uma linguagem para múltiplos bancos de dados chamada SQL*PLUS. Esta linguagem oferece possibilidade de qualificar uma relação com um banco de dados, definir sinônimos para relações ou banco de dados e definir procedimentos entre os bancos de dados envolvidos. Oracle também possui um gerenciador de bancos de dados distribuídos SQL*STAR que permite operações múltiplas que serão avaliadas pelos bancos de dados distribuídos.

Ingres/Star é uma camada de software para acesso transparente ao sistema distribuído Ingres. Permite a definição de um esquema múltiplo externo e virtual através de banco de dados Ingres ou não. Porem, o Ingres/Star não permite que sejam formulados procedimentos nos bancos de dados locais, somente através do esquema externo.

Segundo (GARDARIN; GANNOUNI; FINANCE, 1995) estes produtos podem ser tidos como sistemas de bancos de dados distribuídos *homogêneos*, suportando conexões com SGBDs externos através de gateways. Oracle proporciona SQL*Connect para acessar banco de dados DB2 e outros. Ingres/Star proporciona Gateway-SQL para acessar banco de dados Oracle, e assim por diante. Estas conexões

possuem capacidades limitadas, principalmente no que diz respeito a interoperabilidade de dados.

Multibase (LANDERS; ROSENBERG, 1982), Mermaid (TEMPLETON et. al., 1987), Pegasus (AHMED et. Al, 1991), IRO-DB (GARDARIN; GANNOUNI; FINANCE, 1995) são exemplos de protótipos.

Multibase, utiliza um esquema integrado global e uma linguagem de procedimentos funcional única para manutenção dos dados nos preexistentes bancos de dados autônomos. Multibase suporta definição de dados e uma linguagem de manipulação chamada DAPLEX, a qual é baseada em um modelo de dados funcional. Cada esquema local é traduzido em um esquema expresso em DAPLEX e então integrada ao esquema global. A arquitetura do Multibase é composta por três componentes principais: Gerenciador de Dados Globais (GDG), Interfaces de bancos de Dados Locais (IDL) e SGBD interno. O gerenciador de dados globais manipula os procedimentos globais. Um procedimento global é quebrado em procedimentos individuais nos bancos de dados locais. Estes sub-procedimentos são modificados para compensar as operações que não podem ser expressas nos bancos de dados locais e traduzidos do DAPLEX para a linguagem local, então enviados as suas respectivas interfaces. Quando estes sub-procedimentos são processados os resultados gerados pelos SGBDs locais são traduzidos para o DAPLEX pelo IDL e retornam ao GDG. Os procedimentos executados nos SGBDs locais são direcionados pelo GDG para a devida monitoração de sucesso e/ou erro.

Mermaid, proporciona acesso integrado a bancos de dados heterogêneos e autônomos. Integra bancos de dados relacionais e também pode incluir dados de arquivos. Mermaid não é um SGBD, mas um aplicativo que aloca e integra dados que são mantidos pelos SGBDs locais. Para cada esquema local, Mermaid define um esquema distribuído local. Cada esquema distribuído local representa o subconjunto do banco de dados local que o sistema local compartilha. Os esquemas distribuídos são integrados para definir o esquema global. Todos estes esquemas são

descritos através do modelo de dados relacional. SQL, Quel ou Ariel (a linguagem de consulta com aspectos semânticos) podem ser usadas como linguagem de consulta. A arquitetura do sistema Mermaid consiste de quatro componentes: a Interface do Usuário (IU), o servidor, o dicionário de dados e a interface com os SGBDs. Utilizando a IU, usuários submetem procedimentos globais formulados através de SQL ou Quel ou Ariel. O procedimento global é processado pelo servidor em uma linguagem de dados intermediária (LDI). A LDI gerada é decomposta em sub-procedimentos utilizando o dicionário de dados, o qual contém informações referentes aos bancos de dados envolvidos. O servidor inclui também um otimizador de planos de procedimentos e um controle de execução. Os sub-procedimentos em LDI são transmitidos para as interfaces de SGBDs, as quais os traduzem, executam, recebem os resultados e os transmitem ao servidor. Os componentes do Mermaid utilizam o protocolo TCP/IP para se comunicarem.

Pegasus, proporciona facilidades de acesso e manipulação a múltiplos bancos de dados, sendo eles sistemas autônomos, heterogêneos, distribuídos, orientado a objetos e relacionais, através de uma interface uniforme. Pegasus define um modelo de dados orientado a objetos comum e uma linguagem de dados baseada no modelo objeto de Iri, um banco de dados orientado a objetos desenvolvido pela Hewlett Packard (HP). A linguagem de manipulação de dados é chamada HOSQL (Heterogeneous Object SQL). Pegasus proporciona a facilidade de anexar a um banco de dados Pegasus (banco de dados nativo) outro banco de dados externo (dados estrangeiros), proporcionando assim acesso a múltiplos bancos de dados. Um cliente é conectado a um banco de dados nativo, chamado de banco de dados raiz, através do qual ele pode ter acesso a outro banco de dados anexado a este. Para cada banco de dados estrangeiro, Pegasus associa um banco de dados importado, o qual é considerado como nativo, a única exceção é que seus dados são mantidos em um sistema estrangeiro. A arquitetura do Pegasus consiste de três camadas funcionais. A camada de Acesso a Informação Inteligente (AII) proporciona serviços e interfaces

para comunicação com o Pegasus. A camada de Gerenciamento de Informações Cooperativas (GIC) é responsável pela integração de dados, decomposição e execução de procedimentos e gerenciamento de transações. A camada de Acesso a Dados Estrangeiros (ADE) proporciona os serviços necessários para a tradução dos esquemas estrangeiros no formato Pegasus, para a transformação de procedimentos HOSQL em procedimentos de linguagens nos sistemas estrangeiros e para converter dados estrangeiros em dados Pegasus.

IRO-DB, Interoperable Relacional and Object DataBase, possui um conjunto de ferramentas para realizar interoperabilidade de bancos de dados relacionais pré-existentes e um novo banco de dados orientado a objetos. É baseado em um paradigma orientado a objeto, tanto para descrever bancos de dados heterogêneos quanto para acessar bancos de dados federados. A arquitetura do sistema é divida em três camadas: a camada local avalia os diversos modelos de dados e proporciona uma interface orientada a objetos padrão para cada sistema participante; a camada de comunicação assegura as mudanças das requisições e objetos entre clientes e servidor; a camada de interoperabilidade proporciona visões integradas dos vários bancos de dados participantes e os integra ao meio do sistema IRO-DB.

A abordagem apresentada nos próximos capítulos deste trabalho difere das apresentadas acima por trazer uma solução independente de plataforma, baseada na estrutura XML, e por abranger esta solução ao modelo objeto relacional, incluindo o mapeamento e a decomposição de atributos multivalorados e compostos.

3 A LINGUAGEM SQL PARA O MODELO OBJETO/RELACIONAL

A SQL, do inglês Structured Query Language (Linguagem de Consulta Estruturada), é a linguagem padrão dos bancos de dados relacionais. Foi desenvolvida originalmente na IBM Research, no início da década de 1970. Ela foi implementada pela primeira vez em grande escala em um protótipo da IBM e, depois disso, em numerosos produtos comerciais.

Um banco de dados Objeto/Relacional tem como principal característica admitir recursos tanto do modelo de objetos quanto do relacional, ou seja, é uma tentativa de aproximação entre estas duas tecnologias. Como a base da tecnologia dos principais bancos de dados de hoje é a baseada no modelo relacional, esta aproximação se dá pela evolução dos sistemas relacionais para incorporar alguns recursos dos sistemas de objetos. O mesmo acontece com a linguagem SQL, padrão dos sistemas relacionais, incorpora novos operadores que atendam aos recursos de objeto.

Em sua versão original, era especificamente uma sublinguagem de dados, porém com a inclusão de recursos de armazenamento persistente ao padrão SQL, no final de 1996, a SQL se tornou completa em termos computacionais (DATE, 2000). Hoje, com algumas alterações e conhecida como SQL3, incorpora tipos embutidos, tipos estruturados e herança. Tornando possível, apesar das limitações, a sua utilização em bancos de dados objeto/relacionais.

A SQL, desenvolvida há duas décadas atrás, ainda é uma linguagem linear apesar de seu novo padrão oferecer recursos da orientação a objetos, por isso muitos autores a consideram pobre. Um dos maiores problemas apontado por eles é a forma intuitiva que expressa "*queries*" complexas.

O documento padrão da linguagem SQL, ISO/IEC 9075, possui mais de 600 páginas e as especificações da SQL3 ocupam mais de duas vezes este tamanho. Em consequência, este capítulo está concentrado nos recursos da SQL3 para

manipulação de atributos compostos e multivalorados, suportados pelo modelo objeto/relacional. As definições abaixo seguem um estudo do padrão ISO/IEC 9075 e uma interpretação deste feita por (DATE, 2000).

Entre os novos tipos de dados, está o tipo ARRAY, ele não pode ser definido (não há nenhuma instrução "*create array type*") , ele só pode ser usado através da invocação de uma instrução, "*create table*" por exemplo. Um tipo ARRAY pode ser de qualquer outro tipo, inclusive tipos definidos pelo usuário.

Exemplo 3.1: create table Vendas as

(Item char(5),

Qtde interger array[12])

Neste exemplo o atributo Qtde tem um valor de array (ou matriz). Este atributo consiste em um valor de 12 elementos, cada um do tipo inteiro. Este tipo de dado possui limitações, a principal delas é que os arrays em SQL estão limitados a uma dimensão fixa e seus elementos não podem ser arrays.

O Exemplo 3.2 mostra a manipulação deste tipo de dados.

Exemplo 3.2: Insert into Vendas (Item, Qtde)

Values ('0001', Array[1, 23, 56]);

Update Vendas

Set Qtde[2] = 8

Where Item = '0001';

O tipo ROW é semelhante ao tipo ARRAY no que diz respeito a sua definição, também só pode ser utilizado através da invocação de uma instrução. Um tipo ROW pode ser de qualquer outro tipo, inclusive tipos definidos pelo usuário e pode ser composto por outros atributos.

Exemplo 3.3: create table Vendas as

(Item char(5),

Qtde interger ROW,

Preço ROW (real decimal,

dólar decimal)

Neste Exemplo o atributo Qtde é um atributo inteiro definido como ROW, ou seja, é um atributo multivalorado. O atributo Preço é um atributo do tipo ROW composto por outros dois atributos, real e dólar. O Exemplo 3.4 mostra a manipulação deste tipo de dados.

Exemplo 3.4: Insert into Vendas (Item, Qtde, Preço)

Values ('0001', (1, 23, 56), ((12, 6), (10, 5), (8, 4)));

Update Vendas

Set Qtde[\56] = 60

Where Item = '0001';

Um outro tipo de dado é o tipo estruturado, este deve ser definido pelo usuário através da expressão "*create type*" <nome_tipo> as (<nome_atributos> <tipo_atributos>).

Exemplo 3.3: Create Type Ponto as (X float, Y float)

Create Table Coordenadas as

(Código integer,

Local Ponto);

Neste exemplo o tipo Ponto tem atributos X e Y, os atributos podem ser de qualquer tipo conhecido e não possui nenhum operador diferenciado. Sua sintaxe é a qualificação por ponto, como mostra o Exemplo 3.4. Esta estrutura de

declaração de tipos emprega os atributos compostos definidos no modelo objeto/relacional.

Exemplo 3.4:

```
Insert into Coordenadas                    Insert into Coordenadas
( Código, Local.X, Local.Y )    OU         ( Código, Local )
Values ('0001', 1, 23);                    Values ('0001', (1, 23))

Update Coordenadas
Set Local.X = 8, Local.Y = 50
Where Código = '0001';
```

Este capítulo introduziu a linguagem SQL e apresentou seus recursos para manipulação de atributos compostos e multivalorados, suportados pelo modelo objeto/relacional. No próximo capítulo será abordada a integração de bancos de dados.

4 INTEGRAÇÃO DE BANCOS DE DADOS

Integração de bancos de dados, segundo (SPACCAPIETRA; PARENT, 1994), é um processo que possui como entrada um conjunto de bancos de dados, e produz como saída uma única descrição unificada dos esquemas de entrada, chamado esquema integrado, possui também informações associadas a um mapeamento para dar suporte ao acesso dos dados através do esquema integrado.

A interoperabilidade entre bancos de dados pode acontecer de várias formas (como abordado no Capítulo 2): "*gateways*" comerciais conectam pares de banco de dados; softwares proporcionam facilidades de acesso definindo visões persistentes de diferentes bancos de dados; através de sistemas de bancos de dados distribuídos ou federados.

Até hoje, a mais completa forma de interoperabilidade foi alcançada pelos sistemas de bancos de dados distribuídos ou federados. Banco de dados distribuído é uma coleção de dados espalhados por diferentes locais em uma rede de computador, com diversidade de modelos de dados e SGBDs. Banco de dados federado é uma coletânea de bancos de dados, cujo compartilhamento é feito pelos esquemas de exportação, os quais definem a parte (ou o todo) de cada banco de dados local que será compartilhada. Estas duas abordagem suportam integração de esquemas através de bancos de dados virtuais, ou seja, bancos de dados que são logicamente definidos mas não existem fisicamente. Também permitem que os bancos de dados locais não percam sua autonomia.

O processo de integração de bancos de dados é um processo complexo por ser necessário um intenso conhecimento dos esquemas envolvidos na integração. Os esquemas que serão integrados são chamados de esquemas locais. O objetivo deste processo é definir um esquema conceitual utilizado como interface única dos diversos bancos de dados locais, este esquema único é conhecido como esquema global.

O processo de integração concentra-se na equivalência dos atributos de cada objeto, ou seja, se os atributos que identificam cada entidade ou relacionamento são equivalentes, então o conjunto de entidades e relacionamentos pode ser integrado.

Nas seções 4.1, 4.2 e 4.3 serão apresentadas, segundo (LARSON; NAVATHE; ELMASRI, 1989), as possíveis equivalências entre atributos e objetos e como estas equivalências podem ser exploradas. Na Seção 4.4 será apresentado, como exemplo, o método de integração de esquemas utilizado por (SCOPIM; KATSURAGAWA, 2001) para integrar as diversas bases de dados da Universidade Federal do Paraná.

4.1 EQUIVALÊNCIA DE ATRIBUTOS

Atributos equivalentes possuem várias características em comum, estas características são referenciadas como propriedades de equivalências básicas. Uma equivalência entre atributos pode ser classificada como propriedade de equivalência forte, fraca ou disjunta. O que diferencia uma propriedade de outra é a forma na qual o administrador do banco de dados cria suas funções de mapeamento.

Seguem as definições dos termos acima citados, segundo (LARSON; NAVATHE; ELMASRI, 1989).

Propriedade de equivalência básica: Seja a_i um atributo da classe de objeto A e b_i um atributo da classe de objeto B. Seja D_i um subconjunto não vazio do domínio de a_i, representado por $DOM(a_i)$ e R_i um subconjunto não vazio do domínio de b_i, representado por $DOM(b_i)$, existe equivalência de atributos se existir uma função de mapeamento $f_i : D_i \rightarrow R_i$ e a sua inversa $f_i^* : R_i \rightarrow D_i$.

Função de mapeamento: A função de mapeamento f_i é definida pelo administrador do banco de dados e é utilizada na condição de definir uma correspondência de equivalência entre a_i e b_i. A função f_i deve ser um isomorfismo, ou

seja, deve existir uma correspondência um para um entre D_i e R_i, é esta sua característica que possibilita a existência da função inversa. Para cada operação válida sobre a_i, existe uma operação válida sobre b_i, e vice-versa. A função de mapeamento deve respeitar as regras de integridade, as de segurança e os identificadores únicos.

Exemplo 4.1: *Empregado* é uma classe objeto com três atributos: *número-seguro-social, altura-em-pés* e *grau-escolaridade*. *Professor* é uma classe objeto com três atributos: *número-prof, altura-em-centímetro* e *educação*. Seja f_1 uma função de mapeamento entre *número-seguro-social* e *número-prof*, f_2 uma função de mapeamento entre *altura-em-pés* e *altura-em-centímetro* e f_3 uma função de mapeamento entre *grau-escolaridade* e *educação*. Os atributos *número-seguro-social* e *número-prof* são chaves primárias de *Empregado* e *Professor*, respectivamente, e D_1 representa o domínio e R_1 a extensão da função:

f_1: D_1= DOM (número-seguro-social)

R_1= DOM (número-prof)

$f_1(111.11.111) = 1$

$f_1(222.22.222) = 2$

f_2: D_2 = DOM (altura-em-pés)

R_2 = DOM (altura-em-centímetros)

$f_2(x) = 2.54 * x$

f_3: D_3 = DOM (grau-escolaridade) – {1}

(Razão: o código 1, que representa 1° grau, não existe no domínio DOM (educação))

R_3 = DOM (educação) – {mestrado, doutorado}

(Razão: o atributo educação não tem código correspondente a mestrado/doutorado no domínio DOM (grau-escolaridade))

$f_3(1)$ = indefinido

$f_3(2)$ = 2° grau magistério

$f_3(3)$ = 3° grau

$f_3(4)$ = Pós graduação

No exemplo, as funções de mapeamento são isomórficas. A função f_2, de conversão, é equivalente em ambos altura-em-pés e altura-em-centímetros. As regras de segurança para grau-escolaridade e educação são equivalentes pelo mapeamento de f_3, ou seja, as dependências funcionais são preservadas. Os identificadores únicos são equivalentes através de f_1.

Equivalência de atributo forte: Seja a um atributo (possivelmente multivalorado) de uma classe objeto A e seja b um atributo (possivelmente multivalorado) de uma classe objeto B. Uma equivalência forte entre a e b pode ser criada dependendo do contexto das classes objeto A e B. Uma equivalência forte α relaciona os valores do atributo a em uma instância do conjunto de entidades da classe de objetos A com os valores do atributo b em uma instância de entidades da classe de objetos B, em um determinado momento. A equivalência forte α depende basicamente de D, o subconjunto não vazio do domínio de a, e de R, o subconjunto não vazio do domínio de b, nos quais a função de mapeamento f e sua inversa são definidas. Existem quatro tipos de equivalência forte, que dependem basicamente da quantidade de valores do atributo a que está relacionada com o atributo b.

1) O atributo a relacionado com o atributo b por equivalência FORTE α IGUAL implica na existência de uma correspondência um-para-um entre o conjunto de todos os valores de a e o conjunto de todos os valores de b;
2) O atributo a relacionado com o atributo b por equivalência FORTE α CONTEM implica na existência de uma correspondência um-para-um entre um subconjunto dos valores de a e o conjunto de todos os valores de b, ou seja, o domínio de a contem o domínio de b;

3) O atributo a relacionado com o atributo b por equivalência FORTE α CONTIDO-EM implica na existência de uma correspondência um-para-um entre o conjunto de todos os valores de a e um subconjunto dos valores de b, ou seja, o domínio de a está contido no domínio de b;
4) O atributo a relacionado com o atributo b por equivalência FORTE α INTERSEÇÃO implica na existência de uma correspondência um-para-um entre um subconjunto dos valores de a e um subconjunto dos valores de b.

Da mesma forma que existe uma equivalência forte α que define relações entre atributos em um determinado momento de tempo[4], existe uma equivalência forte β que define relações entre atributos para todos os momentos de tempo. Existem, também, quatro tipos de equivalência forte β:

1) Se a FORTE α IQUAL b em todos os momentos de tempo que A e B existirem, então a FORTE β IQUAL b;
2) Se a FORTE α IQUAL b ou a FORTE α CONTEM b em todos os momentos de tempo que A e B existirem, então a FORTE β CONTEM b;
3) Se a FORTE α IQUAL b ou a FORTE α CONTIDO-EM b em todos os momentos de tempo que A e B existirem, então a FORTE β CONTIDO-EM b;
4) Se a FORTE α IQUAL b ou a FORTE α CONTEM b ou a FORTE α CONTIDO-EM b em todos os momentos de tempo que A e B existirem, então a FORTE β INTERSEÇÃO b.

---

[4] Por momento de tempo entende-se o estado atual do bancos de dados de acordo com os valores de suas instâncias.

Uma equivalência FORTE β permite alterar os valores dos atributos em D e R. Isto é possível porque a função de mapeamento define uma correspondência entre os valores de D e R, possibilitando a conversão de qualquer valor de D no seu valor correspondente em R. Como a função de mapeamento possui uma inversa, qualquer valor em R da função mapeada pode ser convertido no seu valor correspondente no domínio de D. Se o atributo integrado for expresso nos valores de D, então os valores em D podem ser convertidos para os valores em R, e vice-versa.

A possibilidade de alteração é total se a FORTE β IGUAL b, pois a função de mapeamento e sua inversa são totais a respeito do DOM (a) e DOM (b). Quando DOM (a) – D ou DOM (b) – R não são nulos, ou seja, o domínio e o range de valores da função são menores que o valores que a e b podem assumir, a possibilidade de alteração total não existe. Esta situação ocorre quando a FORTE β CONTIDO_EM b ou a FORTE β CONTEM b, respectivamente. Se o domínio que contiver o valor, DOM (a) se a FORTE β CONTEM b ou DOM (b) se a FORTE β CONTIDO_EM b, existir como uma representação física do atributo, então a alteração será possível.

Exemplo 4.2: Sejam CR1, CR2, CR3, CR4 e CR5 atributos com seus respectivos domínios, DOM (CR1) = {1,2,3,4}, DOM (CR2) = {leigo, iniciante, júnior, pleno}, DOM (CR3) = {leigo, iniciante, júnior, pleno, sênior, master}, DOM (CR4) = {júnior, pleno, sênior, master} e DOM (CR5) = {1, 2}. Estes atributos, apesar de possuírem valores numéricos e não numéricos, referenciam hierarquias. A função *f* de mapeamento relaciona estes dois tipos de codificação da seguinte forma: 1 – leigo, 2 – iniciante, 3 – júnior, 4 – pleno, x – sênior, x – master. Utilizando esta função, a FORTE β, analisada sobre CR1, pode ser resumida:

CR1 FORTE β IGUAL CR2, CR1 FORTE β CONTIDO_EM CR3, CR1 FORTE β INTERSERÇÃO CR4 e CR1 FORTE β CONTEM CR5.

Por não haver mapeamento entre todos os valores de todos os atributos analisados juntos, a equivalência entre os 5 atributos é de INTERSEÇÃO.

Equivalência de atributo fraca: Uma equivalência de atributo fraca entre dois atributos a e b pode ser definida somente se uma realocação for permitida, ou seja, se for possível a substituição do atributo a por f(a), sendo f a função de mapeamento. Uma equivalência de atributo fraca não exige que a função de mapeamento tenha inversa.

Exemplo 4.3: Analisando o atributo CR3 do Exemplo 4.2 e seu domínio, DOM (CR3) = {leigo, iniciante, júnior, pleno, sênior, master} e um outro atributo CR6, onde DOM (CR6) = {inexperiente, experiente}, a função de mapeamento f, onde f(leigo) = f(iniciante) = f(júnior) = inexperiente e f(pleno) = f(sênior) = f(master) = experiente, é a função que relaciona CR3 e CR6. Esta função não possui inversa, logo CR3 FRACO β IGUAL CR6.

No Exemplo 4.3, os valores *pleno, sênior* e *master* foram realocados para *experiente*, assim seus valores não podem ser distinguidos. Desta forma se um valor do atributo integrado for alterado de inexperiente para experiente, não será possível determinar qual o valor do atributo CR3. Assim, alterações não serão possíveis a menos que o administrador do banco de dados determine as regras para alteração de objetos similares.

A diferença entre FRACO β IGUAL e FORTE β CONTEM pode ser vista no exemplo abaixo:

Exemplo 4.4: Analisando os atributos CR3 e CR1 do Exemplo 4.2 e seus domínios, DOM (CR3) = {leigo, iniciante, júnior, pleno, sênior, master} e DOM (CR1) = {1, 2, 3, 4}, com a função de mapeamento f, f(leigo) = 1, f(iniciante) = 2, f(júnior) = 3, f(pleno) = 4. Pode ser observado que f relaciona um subconjunto do DOM (CR3) com todos os valores do DOM (CR1), desta forma é possível definir uma função inversa para f, logo CR3 FORTE β CONTEM CR1.

Equivalência de atributo disjunta: Uma equivalência disjunta ocorre quando as regras de dois atributos são as mesmas, porém não é possível definir uma função de equivalência que relacione um atributo em outro, ou seja, é possível

definir uma equivalência disjunta mesmo quando não existe equivalência forte nem fraca.

Exemplo 4.5: Sejam CR7 e CR8 dois atributos, e DOM (CR7) = {leigo, iniciante, júnior, pleno} e DOM (CR8) = {sênior, master} seus domínios. Não existe uma correspondência entre DOM (CR7) e DOM (CR8), entretanto um novo atributo CR9 pode ser criado através da união de CR7 e CR8, ou seja, DOM (CR9) = DOM (CR7) U DOM (CR8). Assim, esta equivalência pode ser denotada por CR7 β DISJUNTA CR8.

4.2 EQUIVALÊNCIA DE OBJETOS

Classes de objetos e atributos podem ser relacionados basicamente seguindo as mesmas regras de integração, ou seja, um par de classes de objetos (assim como um par de atributos) podem ser relacionados através das regras de equivalência, IGUAL, CONTEM, CONTIDO_EM, INTERSEÇÃO e DISJUNÇÃO, as quais também posem ser fortes ou fracas.

A equivalência entre duas classes de objetos é determinada de acordo com a equivalência de seus atributos identificadores (chaves primárias). Segundo (LARSON; NAVATHE; ELMASRI, 1989) segue sua definição:

Equivalência de objetos: Seja A uma classe objeto com chave primária k_a e atributos a_1, a_2, ..., a_n. E seja B uma classe objeto com chave primária k_b e atributos b_1, b_2, ..., b_n. Considere os conjuntos de instâncias de A e B, em um determinado momento de tempo, representados por RWS[5](A) e RWS(B). $T(k_a)$ denota os valores de k_a em todas as instâncias do objeto A e $T(k_b)$ denota os valores de k_b em todas as instâncias do objeto B, para o mesmo momento de tempo. Assim, é possível definir uma função de mapeamento f: D → R, onde D é um subconjunto não

---

[5] RWS – do inglês *Real World Stats* – Estado do Mundo Real

vazio de k_a e R é um subconjunto não vazio de k_b. Utilizando este mesmo mapeamento, a equivalência ρ entre as instâncias das classes objetos A e B pode ser definida, para qualquer momento de tempo.

O tipo de equivalência entre objetos, que dependem basicamente da relação entre k_a e D e entre k_b e R, sendo eles:

1) Se k_a FORTE β IGUAL k_b, então RWS (A) ρ IGUAL RWS (B). Neste caso as classes objeto A e B são integradas em uma única classe AB com chave primária k_a ou k_b, assim RWS (AB) := RWS (A) U RWS (B);
2) Se k_a FORTE β CONTEM k_b, então RWS (A) ρ CONTEM RWS (B). Neste caso a classe objeto A é integrada em uma classe A' e a classe objeto B em uma subclasse B' da classe objeto A', assim RWS (A') := RWS (A) e RWS (B') := RWS (B);
3) Se k_a FORTE β CONTIDO_EM k_b, então RWS (A) ρ CONTIDO_EM RWS (B). Neste caso a classe objeto B é integrada em uma classe B' e a classe objeto A em uma subclasse A' da classe objeto B', assim RWS (A') := RWS (A) e RWS (B') := RWS (B);
4) Se k_a FORTE β INTERSEÇÃO k_b, então RWS (A) ρ INTERSEÇÃO RWS (B). Neste caso uma nova classe objeto AB é criada e as classes objeto A e B são integradas nas classes A' e B', respectivamente, sendo A' e B' subclasses de AB, assim RWS (A') := RWS (A), RWS (B') := RWS (B) e RWS (AB) := RWS (A) U RWS (B);
5) Se k_a β DISJUNTA k_b, então RWS (A) ρ DISJUNTA RWS (B). Neste caso a integração é feita seguindo o item 1 ou o item 4.

A equivalência ρ entre duas classes de objetos se sobrepõem à equivalência β entre quaisquer atributos destes objetos. Esta restrição pode ser apresentada da seguinte forma:

1) Se RWS (A) ρ IGUAL RWS (B), então a única possibilidade de equivalência entre seus atributos é a β IGUAL b;
2) Se RWS (A) ρ CONTEM RWS (B), então a β CONTEM b ou a β IGUAL b;
3) Se RWS (A) ρ CONTIDO_EM RWS (B), então a β CONTIDO_EM b ou a β IGUAL b.

4.3 ESTRATÉGIAS PARA INTEGRAÇÃO DE ATRIBUTOS E OBJETOS

De acordo com as definições apresentadas nas seções 4.1 e 4.2, segundo (LARSON; NAVATHE; ELMASRI, 1989), existem cinco formas de integração de objetos e várias de integração de atributos (levando em consideração a equivalência de objetos)[6]. Neste capítulo são apresentadas as cinco formas de integração de objetos na Figura 1, e algumas estratégias de integração de atributos nas Figuras 2 a 4.

Estratégia 1 – Integrar todos os atributos não disjuntos: Se os atributos a e b são não disjuntos, então integrar a e b em c', exceto nos casos em que a equivalência das classes objeto se sobreponha. Quando duas classes objeto são integradas, os atributos equivalentes são sempre integrados em um atributo c', onde DOM (c') depende do contexto, como indicado na Figura 2. AB' representa uma

---

[6] A equivalência de atributos e classes de objetos disjuntos depende do contexto e a sua integração deve ser determinada pelo administrador do bancos de dados. Por esta razão, para a equivalência disjunta não foi abordada nenhuma estratégia.

classe objeto criada no esquema integrado a partir das classes A e B. As instâncias de AB' derivam de RWS (A) U RWS (B). A' e B' são classes objeto que representam às classes objeto A e B. Vide Figura 2.

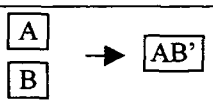
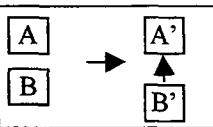
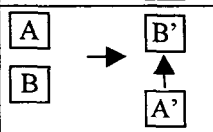
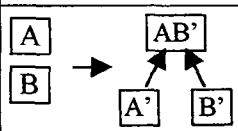
Relação entre RWS (A) e RWS (B)	Integração de A e B
Equivalência RWS (A) ρ IGUAL RWS (B)	 RWS(AB) = RWS(A) U RWS(B)
Inclusão RWS (A) ρ CONTEM RWS (B)	 RWS(A) = RWS(A') RWS(B) = RWS(B')
Inclusão RWS (B) ρ CONTEM RWS (A)	 RWS(A) = RWS(A') RWS(B) = RWS(B')
Interseção RWS (A) ρ INTERSEÇÃO RWS (B)	 RWS(A) = RWS(A') RWS(B) = RWS(B')
Disjunção RWS (A) ρ DISJUNTA RWS (B)	Idem a equivalência ou Interseção

FIGURA 1 – POSSÍVEIS INTEGRAÇÕES ENTRE OBJETOS

Estratégia 2 – Integrar somente os atributos que forem β igual: Se os atributos a e b possuírem equivalência β IGUAL, então integrar a e b em c'. Esta estratégia está dividida em dois casos (ilustrados na Figura 3): a β IGUAL b e NOT (a β IGUAL b). No primeiro caso, a β IGUAL b, a integração ocorre da mesma forma que na estratégia 1. No segundo caso, NOT (a β IGUAL b), assumi-se que a equivalência entre dois atributos significa que estes atributos sejam β IGUAL.

Estratégia 3 – Integrar todos os atributos não disjuntos e migrar valores entre atributos: Como na Estratégia 1, a Estratégia 3 integra os atributos a e b em c' se estes forem não disjuntos. Porém a estratégia 3 permite que valores comuns aos atributos originais apareçam somente uma vez como um atributo integrado. Já os valores que aparecem em somente um dos atributos originais, continuam no atributo original. A Figura 4 ilustra a estratégia e o Exemplo 4.6 mostra a integração de valores.

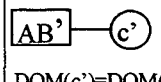
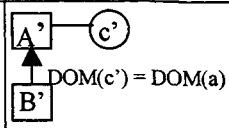
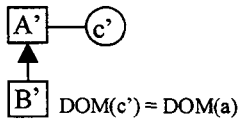
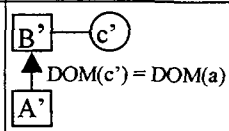
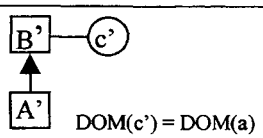
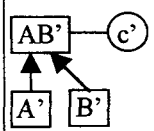
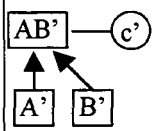
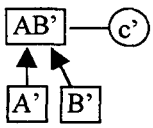
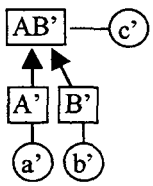
	a β igual b	a β contem b	a β contido_em b	a β interserção b
Equivalência RWS (A) ρ IGUAL RWS (B)	 DOM(c')=DOM(a) or DOM(c')=DOM(b)			
Inclusão RWS (A) ρ CONTEM RWS (B)	 DOM(c') = DOM(a)	 DOM(c') = DOM(a)		
Inclusão RWS (B) ρ CONTEM RWS (A)	 DOM(c') = DOM(a)		 DOM(c') = DOM(a)	
Interseção RWS (A) ρ INTERSEÇÃO RWS (B)	 DOM(c') = DOM(a) or DOM(c') = DOM(b)	 DOM(c')=DOM(a)	 DOM(c') = DOM(b)	 DOM(c') = DOM(a) or DOM(c') = DOM(b) DOM(a') = DOM(a) DOM(b') = DOM(b)
Disjunção RWS (A) ρ DISJUNTA RWS (B)				

FIGURA 2 – ESTRATÉGIA DE INTEGRAÇÃO 1

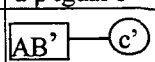
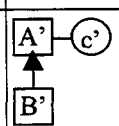
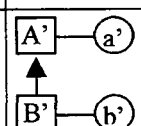
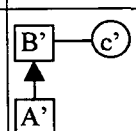
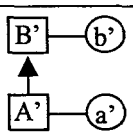
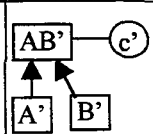
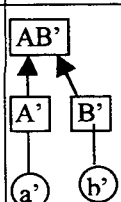
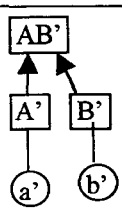
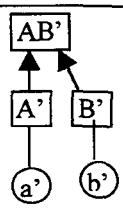
	a β igual b	a β contem b	a β contido_em b	a β interserção b
Equivalência RWS (A) ρ IGUAL RWS (B)				
Inclusão RWS (A) ρ CONTEM RWS (B)				
Inclusão RWS (B) ρ CONTEM RWS (A)				
Interseção RWS (A) ρ INTERSEÇÃO RWS (B)				
Disjunção RWS (A) ρ DISJUNTA RWS (B)				

FIGURA 3 – ESTRATÉGIA DE INTEGRAÇÃO 2

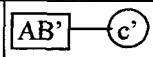
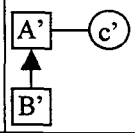
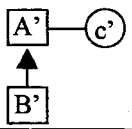
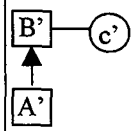
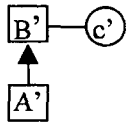
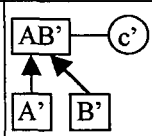
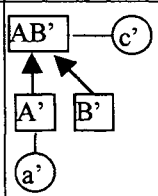
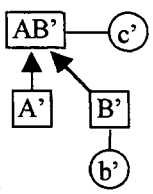
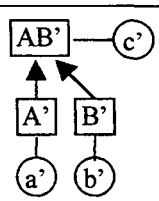
	a β igual b	a β contem b	a β contido_em b	a β interserção b
Equivalência RWS (A) ρ IGUAL RWS (B)				
Inclusão RWS (A) ρ CONTEM RWS (B)				
Inclusão RWS (B) ρ CONTEM RWS (A)				
Interseção RWS (A) ρ INTERSEÇÃO RWS (B)				
Disjunção RWS (A) ρ DISJUNTA RWS (B)				

FIGURA 4 – ESTRATÉGIA DE INTEGRAÇÃO 3

Exemplo 4.6: Considere duas classes de objetos A e B, onde A ρ INTERSEÇÃO B e seus atributos a e b, onde a β CONTEM b. As instâncias de a e b são, respectivamente, $\{x_1, x_2, x_3, x_4\}$ e $\{x_1, x_3, x_5\}$. Como $\{x_1, x_3\}$ são comuns tanto para a quanto para b, na integração eles são migrados para um atributo c' em AB'. As demais instâncias continuam nos seus atributos originais, no caso a' e b'.

4.4 UMA IMPLEMENTAÇÃO DE INTEGRAÇÃO DE BANCOS DE DADOS

O trabalho apresentado por (SCOPIM; KATSURAGAWA, 2001) propõe um mecanismo que permite a junção de vários bancos de dados heterogêneos através da definição de esquemas que representam cada um destes bancos de dados. Estes esquemas são definidos em um único modelo de dados, o que facilita a especificação do esquema integrado. Como modelo único, utiliza a arquitetura

objeto/relacional, pelo seu poder semântico, por unir as facilidades da modelagem tradicional com as capacidades da orientação a objetos.

O sistema da Universidade Federal do Paraná, utilizado como base para o trabalho de (SCOPIM; KATSURAGAWA, 2001), possui bancos de dados heterogêneos gerenciados por diferentes SGBDs e em diferentes plataformas. Estruturado da seguinte forma:

- Sistema de Automação Universitária (SAU). É baseado no modelo hierárquico e utiliza, como SGBD, o DMS-II. Composto por três bancos de dados: 1) Sistema de Protocolo (SAU-01), gerencia documentos ou processos; 2) Gerenciamento de Recursos Humanos (SAU-02), dá suporte a área de RH da UFPR; 3) Controle Acadêmico (SAU-05), informações pessoais e acadêmicas dos discentes da UFPR;
- Sistema de Pesquisa e Pós-Graduação (PRPPG). É baseado no modelo relacional e utiliza SGBD Oracle. Informações sobre o desenvolvimento de pesquisas na UFPR ou de participações oficiais da universidade;
- Sistema Bibliotecário (SIBI). É baseado no modelo relacional e utiliza SGBD MS Access.

A estrutura da Figura 5 foi adota por (SCOPIM; KATSURAGAWA, 2001) por melhorar o tempo de resposta nas consultas via esquema integrado e por não existir uma forma de conexão direta entre os SGBDs Oracle, DMS II e Access.

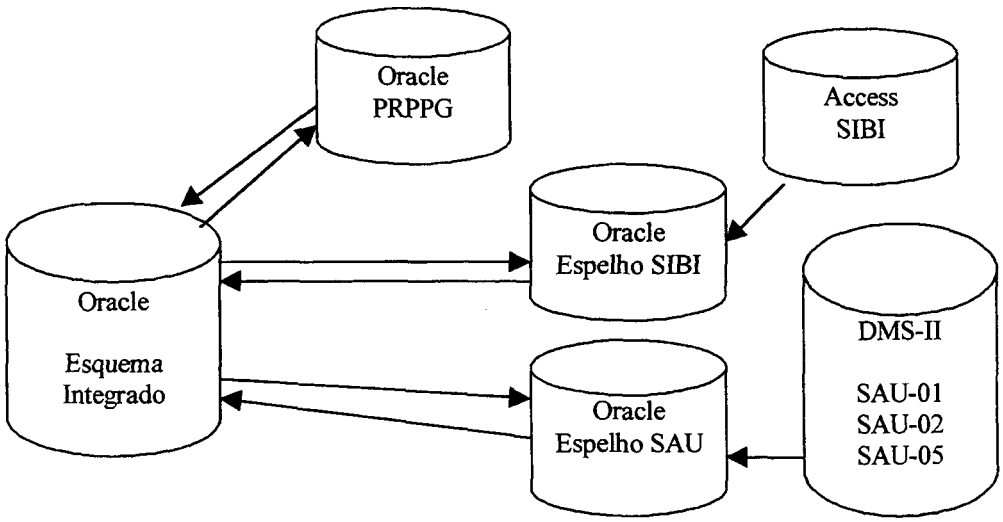


FIGURA 5 – ADAPTAÇÃO PARA CONEXÃO ENTRE ESQUEMAS

A Figura 6 ilustra o resultado final desta integração, onde alguns passos foram essenciais para o desenvolvimento do esquema integrado, são eles:

Passo 1: Pré-integração

Fase onde os esquemas de entrada são transformados para se tornarem mais homogêneos, tanto sintática quanto semanticamente, ou seja, são traduzidos para esquemas utilizando um modelo comum. Esta transformação facilita a especificação do esquema integrado, principalmente quando os esquemas iniciais são projetados em diferentes modelos de dados.

Passo 2: Identificação de Correspondência

Fase onde são feitas a identificação e a descrição dos relacionamentos entre os esquemas. O ponto fundamental é identificar os objetos (entidades, atributos e relacionamentos) nos esquemas iniciais que podem ser relacionados, ou estarem em conflito e classificar as relações entre eles, examinando a semântica dos objetos nos diferentes bancos de dados.

Passo 3: Integração

Fase final, onde são solucionados os conflitos e unificados os itens correspondentes, gerando o esquema integrado. A heterogeneidade identificada através

de objetos conflitantes é resolvida com a ajuda de regras, conhecidas como regras de integração. Cada correspondência (ou conflito) classificada na fase *identificação de correspondência* é amarrada a uma destas regras de integração. São quatro:

Regra de Disjunção: uma entidade genérica é criada com os atributos correspondentes nas duas entidade envolvidas e um relacionamento "*is-a*" com os atributos conflitantes.

Regra de Inclusão: um relacionamento "*may-be-a*" é usado entre as entidades correspondentes.

Regra de Interseção: um relacionamento "*may-be-a*" é usado entre as entidades correspondentes.

Regra de Equivalência: uma entidade é criada para representar a entidade correspondente.

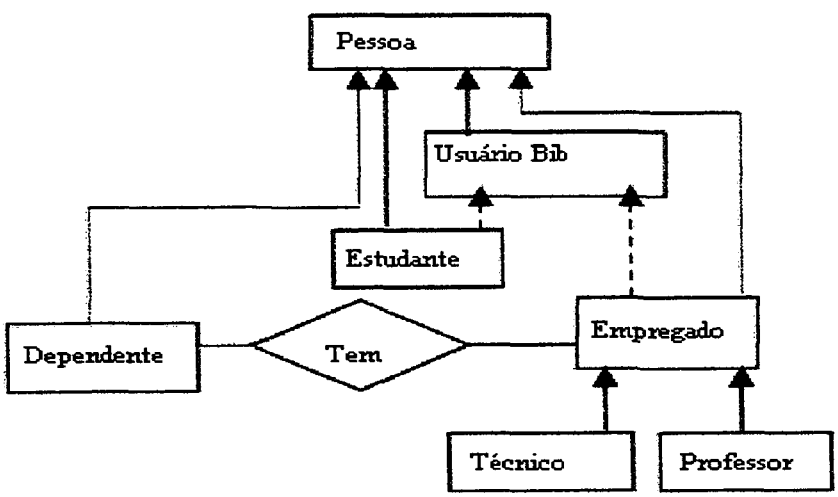


FIGURA 6 – UMA PARTE DO ESQUEMA INTEGRADO

Neste capítulo foi abordado um processo de integração de banco de dados, bem como as possíveis formas de equivalência entre atributos e objetos. No próximo capítulo será apresentado um processo para gerenciamento de transações, utilizado em bancos de dados integrados.

5 GERENCIAMENTO DE TRANSAÇÃO

Uma transação é uma sequência de operações de leitura e gravação, feitas por um usuário, que assegura a consistência, a atomicidade, a durabilidade e o isolamento em um sistema de banco de dados.

Consistência: a execução da transação isoladamente garante a consistência do banco de dados, isto é, se a transação for executada sozinha no sistema;

Atomicidade: ou todas as operações da transação são efetuadas no banco de dados, ou nenhuma;

Durabilidade: todos os valores alterados pela transação permanecem depois da transação ter sido completada corretamente;

Isolamento: o sistema executa somente uma transação por vez.

Em sistemas de bancos de dados, transações de muitos usuários acontecem ao mesmo tempo. Situações indesejáveis podem ocorrer se as operações de vários usuários forem impropriamente intercaladas, ou seja, se uma transação sofrer interferência de outra, infligindo a propriedade de consistência ou isolamento. O controle de concorrência é a atividade que controla a execução das transações para que a interferência de uma com as outras seja aceitável.

Em um sistema de banco de dados integrado, existem transações globais que são executadas sob o controle do SGBD global e transações locais, independentes, que são controladas pelos SGBDs locais. Cada SGBD local integrado pode empregar um gerenciamento de transação diferente e ter completo controle sobre suas transações, inclusive abortar em qualquer ponto da execução da transação. Diante disso, muitas implementações de bancos de dados integrados permitem apenas operações de consulta de dados quando os bancos locais são heterogêneos, pois o controle de concorrência aumenta a complexidade das soluções.

Segundo um modelo apresentado por (BREITBART; MOLINA; SILBERSCHATZ, 1991), o controle de transações em um sistema integrado pode ser feito por dois componentes: um gerenciador de dados global e um gerenciador de transações global. Primeiramente, a transação é decomposta em um conjunto de subtransações pelo gerenciador de dados globais, de acordo com os dados avaliados. Depois o gerenciador de transações global submete e controla a execução de cada uma das subtransações.

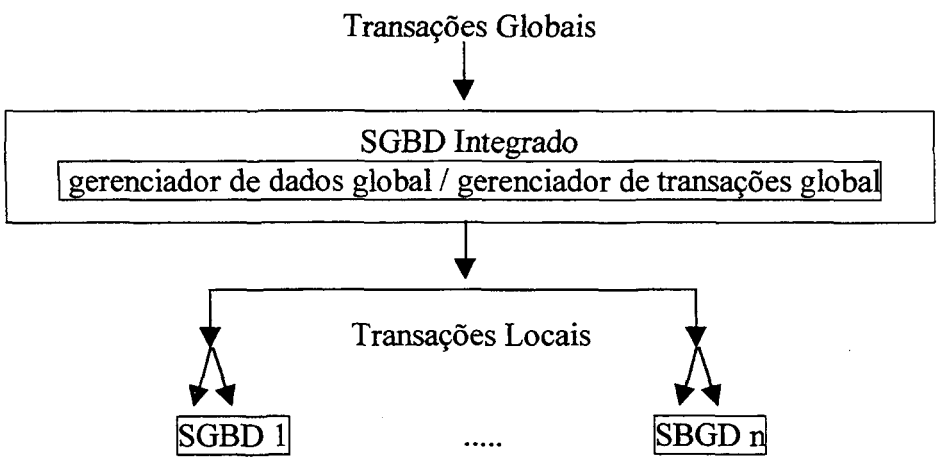


FIGURA 7 – MODELO DE TRANSAÇÕES EM UM ESQUEMA INTEGRADO

Para que seja possível alterações em bancos de dados locais através do banco integrado, a transação global não pode perder suas características básicas – consistência, atomicidade, durabilidade e isolamento. A próxima seção apresenta o modelo básico de transações em bancos de dados integrados e as duas outras subseções apresentam como a consistência, atomicidade, durabilidade e isolamento de uma transação podem ser preservados em um banco de dados integrado, assim como os principais problemas e soluções para que estas característica sejam mantidas.

5.1 MODELO DE TRANSAÇÃO EM SISTEMAS INTEGRADOS

Um sistema integrado é composto por de SGBDs locais e autônomos localizados em espaços físicos, representados s_1, s_2, ...,s_m onde m □ 2. Uma transação T_i, tanto em um sistema integrado quando em um sistema local, consiste de uma operação "*BeginTransaction*[7]", uma coleção parcial de operações para ler e escrever ("*read*[8](x)" e "*write*(x)[9]") e uma operação de "*Commit*[10]" ou "*Abort*[11]". Um sistema integrado suporta dois tipos de transações:

Transações locais: são transações que acessam dados gerenciados por um único SGBD. Estas transações são executadas e controladas pelo SGBD local, independente do controle do SGBD que gerencia o sistema integrado.

Transações globais: são transações que são executadas sob o controle do SGBD integrado. Uma transação global consiste de um número de subtransações, cada uma como uma transação local comum do ponto de vista do SGBD local, onde a subtransação é executada.

Segundo (BREITBART; MOLINA; SILBERSCHATZ, 1991), uma execução no local s_k, denotada por S_k, é uma sequência de transações globais T_i e T_j. Onde inicialmente cada transação é decomposta em subtransações t_{i1}, t_{i2}, ..., t_{in} e t_{j1}, t_{j2}, ..., t_{in}, respectivamente. A decomposição é baseada na localização dos dados que cada subtransação acessa, ou seja, a subtransação carrega as operações e características do SGBD local.

---

[7] Início da transação

[8] Ler

[9] Escrever

[10] Submeter

[11] Recusar

Assim, a notação S_k representa um programa ("*job*") no local s_k, com uma sequência de operações com transações locais e globais resultando em suas execuções. Um "*job*" global S_i é parcialmente ordenado em um conjunto de operações composto por transações globais e locais, tal que, para qualquer local s_k, uma projeção de S_i sobre as transações executadas em s_k é exatamente o mesmo "*job*" S_k no local s_k.

As transações T_i e T_j estão em conflito se existir conflito direto ou indireto entre elas. A transação T_i e a transação T_j estão em conflito direto em S_k se e somente se S_k possuir operações t_{i1}: write(x) e t_{j1}:write(x) consecutivas e T_i não sofrer um "*abort*" antes de T_j executar. A transação T_i e a transação T_j estão em conflito indireto em S_k se e somente se em S_k houver uma sequência de transações T_1, T_2, ..., T_r onde T_i está em conflito direto com T_1, T_1 está em conflito direto com T_2, ..., e T_r está em conflito direto com T_j.

Dois "*jobs*" locais são equivalente se forem definidos com o mesmo conjunto de transações locais e globais, possuírem as mesmas operações e o mesmo conjunto de pares de transação com "*comit*" conflitante.

Todo "*job*" S_k possui um grafo de sequencialização de suas operações, chamado de grafo de serialização local, que é um grafo direcionado com os nós correspondendo para cada transação local ou global que são aceitas, "*commit*", e com ligações do tipo Ti → Tj entre transações em conflito no "*job*" S_k. Um "*job*" S_k pode ser sequencializado se e somente se o seu grafo de serialização local for acíclico.

Um "*job*" S_k possui conflito serial se for equivalente a um outro "*job*" serialmente.

Um "*job*" global é globalizado serialmente se e somente se existir uma ordenação total definida através de transações globais aceitas, com operação "*commit*", que são consistidas com a sequencialização de cada SGBD local.

A união dos grafo de serialização local forma o grafo de serialização global. Um "*job*" global pode ser sequencializado globalmente se e somente se seu grafo de serialização for acíclico.

O Exemplo 5.1 mostra como uma transação local pode provocar conflito indireto entre duas subtransações globais.

Exemplo 5.1: Suponha que as transações globais T_i e T_j executarão operações sobre os itens 'a' e 'b' no local s. E suponha t_{i1} e t_{j1} a decomposição de T_i e T_j, respectivamente, em subtransações:

t_{i1} : read $_{i1}$(a)

t_{j1} : read $_{j1}$(b)

Suponha L uma transação local concorrente as subtransações t_{i1} e t_{j1} executadas em s_k.

L : write(a) write(b)

Sendo s_k o local de execução de L, t_{i1} e t_{j1} e S_k o "*job*" a ser executado. Poderá haver três tipos de sequencialização, que poderão causar ou não conflitos:

"job" 1: Nenhum : Se S_k : write(a) write(b) read $_{i1}$(a) read $_{j1}$(b)

"job" 2: $t_{i1} \rightarrow L \rightarrow t_{j1}$: Se S_k : read $_{i1}$(a) write(a) write(b) read $_{j1}$(b)

"job" 3: $t_{i1} \leftarrow L \leftarrow t_{j1}$: Se S_k : write(a) read $_{i1}$(a) read $_{j1}$(b) write(b)

Embora a ordem de execução de T_i e T_j ser a mesma nos três casos, a transação L interfere na sequencialização de cada "*job*", podendo causar a ocorrência de conflitos.

5.2 CONSISTÊNCIA DE TRANSAÇÕES EM SISTEMAS INTEGRADOS

Como dito anteriormente, o gerenciador de transações global deve garantir as propriedades básicas, citadas acima, da transação global, mesmo na presença de transações locais das quais o gerenciador não é responsável. Além disso, deve ser capaz de garantir a liberação ("*deadlock*[12]") da execução de transações globais e o retorno de erros do sistema.

Nas três subseções seguintes são apresentadas algumas dificuldades que podem contribuir para o não cumprimento da definição acima.

---

[12] Paralisação; Extinção

5.2.1 Sequencialização Global

Existem várias soluções para o problema de sequencialização global em bancos de dados distribuídos homogêneos, estas soluções assumem que cada SGBD local utiliza o mesmo esquema de controle de concorrência e compartilha suas informações. Os vários SGBDs locais podem usar diferentes protocolos para controle de concorrência, por exemplo "*lock*[13]" de duas fases, ordenação por data e hora de agendamento do "*job*", teste de gráfico de serialização, etc. Entretanto, estas soluções não podem ser utilizadas em plataformas de sistemas integrados heterogêneos.

Como transações locais são executadas fora do controle do gerenciador de transações globais, este pode garantir somente a sequencialização global sobre transações globais. Entretanto, em alguns casos, mesmo a execução sequencial das transações globais não garante a sequencialização global. O seguinte exemplo ilustra este fato.

Exemplo 5.2: Seja I um sistema integrado com dois bancos de dados locais localizados em: s_1 com os itens a e b, e em s_2 com os itens c e d. Seja T_1 e T_2 duas transações globais com operações somente de leitura.

T_1: $read_1$ (a) $read_1$ (c)

T_2: $read_2$ (b) $read_2$ (d)

Seja T_3 e T_4 duas transações locais rodando em s_1 e s_2, respectivamente, definidas como:

T_3: $write_3$ (a) $write_3$ (b)

T_4: $write_4$ (c) $write_4$ (d)

Assumindo que a transação T_1 é executada e aceita, operação "*commit*" , nos dois locais e depois a transação T_2 é executada e aceita, operação "*commit*", nos dois locais.

---

[13] Bloqueio

Estas execuções podem resultar nos seguintes "*jobs*" locais S_1 e S_2, gerados em s_1 e s_2, respectivamente:

S_1: $read_1$ (a) $commit_1$ $write_3$ (a) $write_3$ (b) $commit_3$ $read_2$ (b) $commit_2$

S_2: $write_4$ (c) $read_1$ (c) $commit_1$ $read_2$ (d) $commit_2$ $write_4$ (d) $commit_4$

Como resultado, a transação T_1 é sequencializada antes de T_2 em s_1 e depois de T_2 em s_2; desta forma a sequencialização global não é mantida.

No Exemplo 5.2, o problema acontece porque a transação local cria conflitos indiretos entre as transações globais. Como o gerenciador de transações globais não influencia nas transações locais, também não influencia nestes conflitos indiretos. Este fato é a causa da maior dificuldade em tentar assegurar a sequencialização global em um sistema integrado.

5.2.2 Atomicidade Global e Retorno de Informações

A atomicidade global dita que, ou todas as subtransações de uma transação são executadas com sucesso, "*commit*", ou todas elas são recusadas, "*abort*".

Segundo (BREITBART; MOLINA; SILBERSCHATZ, 1991), em sistemas de bancos de dados integrados homogêneos, a atomicidade das transações é assegurada por um protocolo que requer que os bancos de dados locais tenham um *estado de preparação* para cada subtransação. A subtransação pode permanecer no estado de preparação até que o gerenciador decida se deve sofrer "*commit*" ou "*abort*".

Porém, para que a autonomia dos bancos de dados locais participantes seja preservada, os bancos de dados locais não podem estar condicionados a decisão do gerenciador global para aceitar ou abortar suas transações. Ou seja, a exportação do estado de preparação de suas transações não pode ser obrigatória. Isto não somente mostra que a transação global não é atômica, mas também causa incorreções nos "*jobs*" globais, como mostra o Exemplo 5.3:

Exemplo 5.3: Seja I um sistema integrado com dois bancos de dados locais localizados em: s_1 com o item a, e em s_2 com o item c. Seja T_1 a seguinte transação global:

T_1: $read_1$ (a) $write_1$ (a) $write_1$ (c)

Tendo T_1 completado suas ações de leitura e escrita em ambas as localizações e o gerenciador de transações globais enviado a confirmação de "*commit*" também para ambas, s_2 recebe o "*commit*" e então aceita suas subtransações. Entretanto, s_1 decide abortar suas subtransações antes que o "*commit*" chegue, não executando a transação T_1. Depois disso, uma transação local T_2 é iniciada em s_1:

T_2: $read_2$ (a) $write_2$ (a)

Neste ponto, o resultado global do "*job*" está incorreto, pois só completou metade das operações de T_1. Para acertar a situação, o gerenciador de transações globais, submete novamente a s_1 suas operações, ou seja, $write_1$ (a). O SGBD local em s_1 considera esta operação como uma nova transação T_3 não relacionada a T_1. Desta forma, para s_1 existe o seguinte "*job*":

S_1 : $read_2$ (a) $write_2$ (a) $write_3$ (a)

No Exemplo 5.3, se o SGBD local tivesse preparado a operação antes de executá-la, "*prepare-to-commit*[14]", os problemas do exemplo poderiam ter sido evitados. Neste caso, o gerenciador de transações global não teria emitido a ação de "*commit*" para T_1 antes que s_1 e s_2 retornassem o estado da ação "*prepare-to-commit*". Como já dito anteriormente, esta solução viola a autonomia de execução de um SGBD.

Existem várias soluções propostas para que sistemas integrados utilizem as vantagens de preparar a operação antes de executá-la, sem que os sistemas locais percam sua autonomia. Uma destas soluções é um protocolo para "*commit*" de

[14] Preparar-para-aceitar

duas fases, que utiliza a operação "*prepare-to-commit*", nesta solução todos os SGBDs devem dispor deste serviço.

5.2.3 Paralisação ("*deadlock*") Global

Em um sistema de banco de dados integrado cada SGBD local utiliza um mecanismo próprio de bloqueio, "*lock*", para assegurar a sequencialização local e para detectar e retornar de um "*deadlock*". Entretanto, em um sistema integrado há a possibilidade de um "*deadlock*" global e este não ser detectado pelo gerenciador de transações global.

Exemplo 5.4: Seja I um sistema integrado com dois bancos de dados locais localizados em: s_1 com os itens a e b, e em s_2 com os itens c e d. Os dois bancos de dados locais utilizam "*lock*" de duas fases para garantir a sequencialização local. Seja T_1 e T_2 duas transações globais com operações somente de leitura.

T_1: $read_1$ (a) $read_1$ (d)

T_2: $read_2$ (c) $read_2$ (b)

Seja T_3 e T_4 duas transações locais rodando em s_1 e s_2, respectivamente, definidas como:

T_3: $write_3$ (b) $write_3$ (a)

T_4: $write_4$ (d) $write_4$ (c)

Tendo T_1 executado a operação $read_1$ (a) e T_2 executado a operação $read_2$ (c), logo após, em s_1, a transação local T_3 executou $write_3$ (b), submeteu a operação $write_3$ (a) e foi forçada a esperar pela finalização do "*lock*" no item a mantido por T_1. Em s_2, a transação T_4 executou $write_4$ (d), submeteu a operação $write_4$ (c) e foi forçada a esperar pela finalização do "*lock*" no item c mantido por T_2. Finalmente, as transações T_1 e T_2 submeteram suas próximas operações, $read_1$ (d) e $read_2$ (b), e o "*deadlock*" aconteceu.

Por serem autônomos, os SGBDs locais não podem alterar seu controle de informação e portanto não percebem que estão em um "*deadlock*" global.

Assim como, o SGBD integrado não reconhece as transações locais e, portanto, também não reconhece o "*deadlock*" global. Uma solução simples e independente de controle, para que os sistemas saiam do "*deadlock*" global, é o "*timeout*", ou seja, um tempo definido estrategicamente para que o sistema saia da posição de espera.

5.3 ATOMICIDADE E DURABILIDADE DE TRANSAÇÕES EM SISTEMAS INTEGRADOS

Em um sistema de banco de dados integrado, assim como em um banco de dados único, falhas podem causar recusa "*abort*" de uma transação. Estas falhas podem ser do sistema, como inconsistência de dados, ou do gerenciador de transação global, como interligação ou comunicação entre os SGBDs locais. Assim como, uma transação global em um local s_k pode sofrer um "*abort*" decorrente de uma operação do SGBD local (como "*abort*" causado pela detecção de "*deadlock*") e a mesma transação sofrer um "*commit*" em outro local.

Procedimentos de recuperação em bancos de dados integrados devem assegurar que o gerenciador de transações global possa recuperar informação tanto de recusas unilaterais, provenientes dos SGBDs locais, como de falhas. Desde que os procedimentos de recuperação, em cada SGBD local, assegurem a atomicidade e a durabilidade de transações locais e subtransações globais, a tarefa de garantir atomicidade e durabilidade para transações em sistemas integrados se reduz a assegurar que cada transação global ou executa "*commit*" para todas as suas subtransações ou "*abort*" para todas elas.

Além dos procedimentos de recuperação, existem outros mecanismos para garantir a atomicidade e durabilidade de uma transação global. Como os apresentados abaixo:

Refazer: as operações de escritas da subtransação que falhou são colocadas em nova transação para serem executadas novamente. Esta transação é composta por todas as operações de escrita executadas pela subtransação.

Tentar novamente: toda a subtransação que falhou, e não somente suas operações de escrita, são executadas novamente.

Compensação: em cada local que a subtransação de uma transação global foi executada com sucesso, ou seja, não falhou, uma transação de compensação é executada para semanticamente desfazer seus efeitos.

Enquanto as técnicas refazer e tentar novamente asseguram a atomicidade de uma transação, no caso da técnica de compensação uma notação mais fraca de atomicidade é usada, desde que seja possível que os efeitos da transação global recusada, "*abort*", sejam transferidos para outra transação. Isto impacta na preservação da consistência de um sistema.

5.3.1 "*Commit*" de Duas Fases

O "*commit*" de duas fases, 2PC[15], age da seguinte forma: no término da execução das operações de transações, o gerenciador de transação global submete uma operação "*prepare-to-commit*" a cada SGBD local onde a transação é executada. Ao receber a operação "*prepare-to-commit*", o SBDG local vota para aceitar, "*commit*", ou rejeitar, "*abort*", a transação. Se votar para que a transação seja aceita, "*commit*", ele entra no estado de preparação para a transação. Ao entrar no estado de preparação, o SGBD local cede seu direito de rejeitar a transação, "*abort*", ao gerenciador de transação global. O gerenciador, ao receber os votos de cada local onde a transação foi executada, dependendo dos votos, decide por um "*commit*" ou um

[15] 2PC – do inglês two phase commit – aceite de duas fases.

"*abort*" da transação. Todos os SGBDs locais em estado de preparação aceitam a decisão do gerenciador de transações global.

Para que o "*commit*" de duas fases funcione, é preciso que todos os SGBDs locais envolvidos entrem em um estado de preparação, ou seja, fiquem em posição de executar um "*commit*" ou "*abort*" na transação de acordo com a instrução do gerenciador (inclusive na presença de falhas). Desde que seja possível ocorrer uma queda total do sistema enquanto o SGBD local estiver no estado de preparação, ele deve guardar as alterações feitas pela transação antes de entrar neste estado. Por outro lado, desde que seja possível outras transações sofrerem um "*abort*" enquanto o local estiver em estado de preparação, ele deve assegurar que este "*abort*" não comprometa seu consentimento com a decisão do gerenciador de transações global.

Para permitir estas duas situações, antes de entrar no estado de preparação da transação T_i, o local s_k, deve assegurar que em cada transação T_j da qual T_i tenha lido um item de dado em s_k, seja feito um "*commit*". Se não, é possível que o gerenciador de transações global decida por um "*commit*" em T_i, mas como T_i leu um dado em s_k escrito por T_j, e T_i sofreu um "*abort*", T_i não pode mais ter um "*commit*" no SGBD local s_k. Se o SGBD local produzir a sequencialização das suas operações, então a propriedade acima é assegurada.

Existem algumas situações onde o "*commit*" de duas fases não é indicado, por exemplo:

- SGBDs locais que dão a clientes remotos um conjuntos de serviços, mas não controlam o "*commit*" destes serviços;
- SGBDs locais que desejam reter sua autonomia de execução e comunicação;
- Quando a performance do "*commit*" de duas fases em um sistema distribuído for inadequada. Em particular, os SGBDs ficam retidos no estado de preparação por algum tempo, bloqueando recursos locais, o tempo de resposta das transações pode ser afetado.

Este capítulo apresentou um modelo básico de gerenciamento de transações em bancos de dados integrados. O próximo capítulo apresenta a linguagem de marcação XML, suas características e vantagens pelas quais foi utilizada neste trabalho.

6 LINGUAGEM DE MARCAÇÃO XML

Em 1996, a World Wide Web Consortium, ou W3C, apresentou o primeiro rascunho ("*draft*") da linguagem de marcação de textos estendida, XML, que é uma adaptação para a Web de uma linguagem de marcação padrão de uso geral, a SGML[16], que foi desenvolvida na IBM em 1969, com a finalidade de estruturar documentos legais.

XML é uma tecnologia que cresceu rapidamente e foi envolvida pela segunda geração da WEB, seguindo a revolução iniciada pela combinação de hipertextos e Internet globalizada. Sua maior contribuição foi a inclusão de significado na WEB, possibilitando que os dados carregassem não somente informações estruturais mas também conteúdo. Assim, tornou-se uma grande facilidade em processos de informação.

Embora criada como um formato padrão para troca de dados estruturados na WEB, a XML se tornou uma grande ferramenta em outros meios. Entre seus grandes méritos, está o seu potencial em prover a possibilidade de interoperabilidade de estrutura e conteúdo entre sistemas distribuídos. A estrutura e o tipo de informação de um documento XML podem ser precisamente armazenadas em arquivos de definição do documento, DTD[17], os quais podem ser passados, juntamente com o documento XML, de uma aplicação para outra. O formato DTD também é padronizado, não havendo ambiguidade entre a sua sintaxe e a do documento XML.

Sendo um formato de dados auto descritivo, baseado em texto e universal, XML é uma excelente escolha para representação de dados ou objetos.

---

[16] SGML – do inglês *Standard General Markup Language* – Linguagem de marcação padronizada de uso geral

[17] DTD – do inglês *Document Type Definition* – Definição de Tipos do Documento

Sendo uma meta linguagem, pode ser utilizada para definir outros domínios ou linguagens específicas de um domínio (ex.: matemático, químico, biológico).

A grande expansão da XML foi devido a algumas de suas características, entre elas:

- Ser aberta. XML pode ser utilizada para compartilhar dados com outros usuário e programas em plataformas independentes;
- Ser naturalmente auto descritiva, eliminando a necessidade de conhecimento prévio do modelo;
- Possuir extensões e vocabulários específicos para cada domínio;
- Ser independente de plataforma, pois descreve dados e não apresentação visual.

Para trabalhar com XML a W3C criou alguns padrões de acesso, resumidamente descritos abaixo:

- XML – a linguagem em si;
- XLL (Xlink and Xpointer) – para ligação entre documentos e partes de documentos, hipertextos;
- XSL (Extensible Style Language) e XSLT (Extensible Style Language Transformation) – permitem a transformação de um documento em outro, para fins de apresentação ou intercâmbio;
- DOM (Document Object Model) – estrutura criada com a finalidade de possibilitar a navegação, escrita e leitura, em documentos XML;
- XML NameSpaces – para resolver conflitos de nomes;
- XML Schemas – para especificação de tipos de dados, subclasses e melhores modelos de conteúdo;
- XML Query – permite exprimir consultas SQL em banco de dados.

A XML não representa somente um padrão para formatar dados, mas também é destinada para prover interoperabilidade e processamentos automáticos para conhecimento de dados. Através dela é possível descobrir a estrutura de um dado, seu tipo ou semântica.

As subseções seguintes apresentam a sintaxe da XML e a estrutura do documento DTD.

6.1 A SINTAXE XML

Um documento XML consiste de três partes, como mostra a Figura 8. O prólogo onde são declarados os nome de elementos, atributos e as regras para validar as marcações. O corpo do documento, que possui dados com marcação, consiste de um ou mais elementos em forma de árvore hierárquica e é composto basicamente por elementos e caracteres de dados. O epílogo, que é uma miscelânea de comentários, instruções de processamento e/ou espaços em branco que seguem o final da árvore de elementos. O prólogo e o epílogo são opcionais.

Quanto a marcação, é feita através de uma marca, conhecida como TAG. Tags são códigos incorporados ao texto do documento que armazenam informações necessárias para o processamento eletrônico, como nome de fonte ou negrito, no caso da XML as tags são utilizadas para demarcar o início e o fim de um elemento, delimitam a estrutura do documento.

Prólogo

```
<?xml version="1.0"?>
<!-comentários e instruções de processamento>
<!DOCTYPE livros SYSTEM http://meucomputador/livros.dtd>
<!-comentários e instruções de processamento>
```

Corpo

```
<livros>
        <livro categoria = "romance">
            <autor>Jorge Amado</autor>
            <titulo>Gabriela Cravo e Canela<titulo>
            <preço>50,00</preço>
        </livro>
        <livro categoria = "infantil">
            <autor>Monteiro Lobato</autor>
            <titulo>Sítio do Pica Pau Amarelo<titulo>
            <preço>35,00</preço>
        </livro>
</livros>
```

Epílogo

```
<!-comentários e instruções de processamento>
```

FIGURA 8 – EXEMPLO DE DOCUMENTO XML

6.1.1 O Prólogo de um Documento XML

Um documento XML começa com o prólogo. O prólogo é utilizado para sinalizar o início dos dados XML, descrever o método de codificação de caracteres e prover outras informações necessárias ao interpretador do documento ("*parser*[18]") e às aplicações.

Na primeira linha do prólogo estão as informações que indicam o início do documento e as informações necessárias para sua validação, conforme a sintaxe <?XML Version Enconding RequiredMarkeupDeclaration ?>, na qual <?XML é um marcador de início, obrigatório, Version refere-se a versão da recomendação do W3C, Enconding representa o conjunto de caracteres a ser utilizado e

[18] Tradutor

RequiredMarkeupDeclaration determina se o documento poderá ser validado somente com as informações internas ou não, isto é, se é necessário um arquivo DTD ou esquema XML e ?> o finaliza a declaração (MARTIN, et al., 2000).

A próxima linha significativa, excluindo linhas de comentários, trás informações referentes ao documento em si, como seu tipo, nome e a localização de suas regras de validação, conforme a sintaxe <!DOCTYPE RootElement SYSTEM ExternalDeclaration [InternalDeclaration]>, onde <!DOCTYPE representa o início da declaração do tipo do documento, RootElement aponta o elemento raiz e SYSTEM indica as regras de validação. No caso de validação externa, ExternalDeclaration indica o nome e localização do documento de validação agregado, no caso de validação interna, as regras serão declaradas na sequência e contidas entre parênteses (MARTIN, et al., 2000). Como mostra a seção prólogo da Figura 8.

6.1.2 O Corpo do Documento XML

O corpo de um elemento XML encontra-se logo após as declarações do prólogo, é formado por um ou mais elementos e pode também conter caracteres de dados.

Elemento é um nome conceitual. Os elementos são a construção básica no formato de blocos de uma marcação XML e podem conter outros elementos ou textos, que são chamados de conteúdo de um elemento. Enquanto marcadores, utilizam tags, nas quais consistem o nome do elemento entre os símbolos "<" e ">". Todo elemento deve ser delimitado por uma tag-início e uma tag-fim. Como no exemplo da Figura 8, o elemento <autor>Jorge Amado</autor> é composto pela tag-iníco <autor>, pelo conteúdo Jorge Amado e pela tag-fim </autor>.

Elementos podem possuir informações adicionais que os qualificam. Estas informações podem ser anexadas aos elementos na forma de atributos, os quais são compostos pelo par nome-valor. Onde o valor de um atributo poderá ser nulo,

assumir um valor discreto ou possuir um conjunto infinito de valores. Os atributos sempre ocorrem dentro da tag inicial. Um exemplo de elemento com atributo é encontrado na Figura 8, <livro categoria = "romance">, onde o elemento livro possui o atributo categoria com o valor romance. Os valores de atributos sempre serão indicados entre aspas, independente deles serem numéricos ou não.

Caracteres de Dados é qualquer texto que faz parte do conteúdo de um elemento ou atributo. Caracteres de marcação não fazem parte deste conjunto, por exemplo, o caracter & ou o caracter < são caracteres delimitadores, por isso nunca poderão aparecer no seu formato literal no corpo de um texto. Se houver a necessidade destes caracteres aparecerem no conteúdo de um elemento deverá ser usada uma referência, &amp para o caracter & e &lt para o caracter <. Estas referências estão definidas na especificação da XML

6.2 O DOCUMENTO DE DEFINIÇÃO DE TIPO – DTD

Quando programas são desenvolvidos tanto para gerar documentos XML quanto para utilizá-los, as intenções do designer estão implícitas no código – este elemento segue aquele elemento, o qual significa isto, simplesmente porque o designer definiu assim. Entretanto, se houver a necessidade de um documento XML gerado por um programa ser utilizado por outros, deverá haver um caminho comum para o designer de um vocabulário XML comunicar as regras sintáticas deste vocabulário para que qualquer outro usuário possa interpretar a estrutura do documento, ou seja, este elemento segue aquele elemento porque está especificado aqui.

Diante desta necessidade, foi especificado um mecanismo como parte da estrutura XML, o documento de definição de tipos, DTD. O DTD utiliza uma gramática formal para especificar a estrutura e os valores permitidos de um documento XML. O papel do documento DTD é definir todos os elementos de um documento

XML, os atributos que podem estar associados a cada um destes elementos e as relações entre os elementos.

O conteúdo válido de um documento DTD é definido em termos de quatro marcadores básicos, pré definidos, conhecidos como construtores DTD. Sendo eles apresentados abaixo segundo (MARTIN, et al., 2000):

ELEMENT[19]	Declaração de um elemento XML.
ATTLIST[20]	Declaração dos atributos que podem ser associados a um elemento específico e os valores permitidos a esse atributo.
ENTITY[21]	Declaração de conteúdo reutilizado.
NOTATION[22]	Declaração de formatos de conteúdos externos que não serão validados (dados binários, por exemplo).

Nas próximas subseções serão apresentados cada um destes marcadores.

6.2.1 Entidades

Entidade é a facilidade de declarar conteúdos e referenciá-los quantas vezes for necessário. Com a declaração de uma entidade no DTD é definido o nome e o conteúdo a ser referenciado.

A declaração de uma entidade é feita através da palavra chave ENTITY, seguida pelo nome da entidade e os valores que serão substituídos. Por

---

[19] Elemento

[20] Lista de Atributo

[21] Entidade

[22] Notações

exemplo, <!ENTITY booleano "(sim | não")>, onde booleano é o nome da entidade e "(sim | não)" seu valor.

Com esta declaração, toda vez que for necessário escrever "(sim | não)" coloca-se a palavra booleano como referência no lugar. Para que o nome de uma entidade não seja confundido com os marcadores, ele é delimitador pelo caracter '&' a sua frente e pelo caracter ';' no final, (por exemplo, &booleano;).

6.2.2 Elementos

Elementos são declarados no DTD através da palavra chave ELEMENT. Possuem um nome e a especificação do seu conteúdo. Os elementos são divididos em quatro categorias: vazios, elemento, mistos e qualquer.

Vazio: um elemento vazio é um elemento que não possui texto nem é composto por elementos filhos. É denotado pela palavra chave EMPTY.

Elemento: na sua melhor terminologia, esta categoria significa que ele é somente elemento, ou seja, é composto somente por elementos filhos, não possui texto.

Misto: um elemento misto é um elemento composto por elementos filhos e caracteres de dados (#PCDATA), texto.

Qualquer: esta denominação é utilizada para indicar que um elemento possui qualquer conteúdo sem violar a sintaxe XML.

Elemento vazio e qualquer são declarados da seguindo a sintaxe: <!ELEMENT SemDados EMPTY> e <!ELEMENT QualquerDado ANY>. Existem vária razões para a definição de um elemento vazio, a mera existência deste elemento pode significar alguma coisa no documento como um todo. Um elemento qualquer é genérico e pode conter qualquer combinação de elementos textos.

As categorias elemento e misto podem usar estruturas para expressar significado, são indicadas como modelo de conteúdo. Um modelo de conteúdo é a

especificação da estrutura interna do conteúdo de um elemento, consiste de um conjunto de parênteses que envolve uma combinação de nome de elementos filhos, operadores e a palavra chave #PCDATA. Os operadores são usados para denotar cardinalidade e indicar como os elementos e caracteres de dados (#PCDATA) são combinados.

Elementos podem ser combinados em sequência, através do operador de ordem ',' (virgula) ou pela escolha de um valor, operador '|' (pipe). Como mostra a Figura 9.

```
<!ELEMENT Nome (pnome, snome )>
<!ELEMENT Fruta (maçã | laranja )>
<!ELEMENT CestaFruta ( limão, (maçã | laranja )>
```

FIGURA 9 – FRAGMENTO DE UM ARQUIVO DTD

Na Figura 9 os elementos pnome, snome devem aparecer e devem estar na ordem especificada. Na especificação do elemento Fruta, ele pode conter o elemento maçã ou laranja, nunca os dois. Já na especificação do elemento CestaFruta, ele deve conter o elemento limão seguido do elemento maçã ou do elemento laranja. Como mostra a Figura 10.

```
<CestaFruta>
                <limão> ... </limão>
                <maçã> ... </maçã>
</CestaFruta>

<CestaFruta>
                <limão> ... </limão>
                <laranja> ... </laranja>
</CestaFruta>
```

FIGURA 10 – INSTÂNCIA DE ELEMENTOS EM XML

A quantidade de instâncias de um elemento pode ser denotada através dos operadores de cardinalidade. O operado '?' (interrogação) indica que o elemento é opcional, ou seja, pode ou não aparecer, o operador '*' (asterisco) indica que o elemento pode aparecer nenhuma ou várias vezes e o operador '+' (sinal de mais) indica que o elemento aparece uma ou várias vezes.

```
<!ELEMENT PacoteMisto (#PCDATA | ItemA | ItemB)*>
        <PacoteMisto>
                        <ItemA> ... </ItemA>
                        <ItemB> ... </ItemB>
                        texto qualquer como exemplo de #pcdata
                        <ItemB> ... </ItemB>
                </PacoteMisto>

<!ELEMENT Pacote (ItemA+ | ItemB )>
        <Pacote>
                <ItemA> ... </ItemA>
                <ItemA> ... </ItemA>
                <ItemB> ... </ItemB>
        </Pacote>
```

FIGURA 11 – ELEMENTOS EM DTD E SUAS INSTÂNCIAS

6.2.3 Atributos

Atributos complementam e modificam elementos através da associação de propriedades. Permitem a inclusão de uma grande quantidade de informações. Cada elemento que possuir atributos declarados terá pelo menos uma declaração com a palavra chave ATTLIST, seguida do nome do elemento, ao qual a lista de atributos será anexada, mais a definição dos atributos. A definição do atributo consiste de um nome, tipo e um valor padrão ("*default*").

Há quatro valores padrão para a declaração de um atributo:

Requerido: palavra chave #REQUIRED. O atributo deve aparecer em toda instância do elemento.

Significativo: palavra chave #IMPLIED. O atributo pode aparecer opcionalmente na instância de um elemento.

Fixo: palavra chave #FIXED mais valor padrão. O atributo deve sempre ter o mesmo valor. Se o atributo não aparecer, seu valor é assumido.

Valor padrão: não possui palavra chave, somente valor. Se o atributo não aparecer, o valor padrão é assumido. Se o atributo aparecer, ele pode conter um outro valor.

Há algumas categorias que definem o tipo de um atributo, cada uma delas possui uma palavra chave para identificá-las. Segue abaixo suas especificações:

CDATA: texto. Sequência de caracteres.

ID: Nome único em um dado documento.

IDREF: Referencia algum elemento que carrega um atributo do tipo ID que possui o mesmo valor que o atributo IDREF.

IDREFS: Série de IDREF delimitados por espaço em branco.

ENTITY: Nome de uma entidade externa pré-definida.

ENTITIES: Série de nomes de ENTITY delimitados por espaço branco.

NMTOKEN: Um nome.

NMTOKENS: Série de NMTOKEN delimitados por espaço branco.

NOTATION: Aceita um nome ou um conjunto de nomes de notações declaradas no DTD.

[Valores Enumerados]: Aceita um ou uma série de valores definidos pelo usuário que o atributo pode ter.

```
<!ATTLIST livro
                ISBN        ID              #REQUIRED
                Categoria   CDATA           "profissional"
                DtPublic    CDATA           #REQUIRED
                NumPag      CDATA           #REQUIRED
                Autores     IDREFS          #IMPLIED
                Doação      (sim | não)     #REQUIRED
>
```

FIGURA 12 – EXEMPLO DE DECLARAÇÃO DE ATRIBUTOS

Como mostra a Figura 12, o elemento livro possui vários atributos. O atributo ISBN, por exemplo, é um atributo do tipo ID obrigatório (requerido), ele indica que para toda instância do elemento livro, haverá um atributo ISBN com valor único. O atributo Categoria é um atributo do tipo texto com o valor padrão "profissional", se ele não aparecer na instância do elemento indica que o livro é da categoria profissional, caso contrário, aparece na instância e indica outro valor. Os atributos DtPublic e NumPag são do tipo texto e obrigatórios em todas as instâncias do elemento livro. O atributo Autores é do tipo IDREFS e opcional, indica que para cada valor contido no atributo Autores do elemento livro existe um atributo do tipo ID em um outro elemento com o mesmo valor.

Os tipos de atributo ID, IDREF e IDREFS possibilitam expressar relacionamentos normalmente encontrados em bancos de dados relacionais.

Ainda tendo como exemplo a Figura 12, o atributo Doação é do tipo enumerado e obrigatório. Atributos do tipo enumerado possuem uma lista de valores, dos quais um e somente um poderá ser tido como valor do atributo na instância do elemento. Ou seja, o atributo Doação poderá conter somente o valor 'sim' ou o valor 'não'.

A Figura 13 mostra um documento DTD completo. Como apresentado nesta seção, documentos DTD forçam os documentos XML a terem uma estrutura precisa e sem ambiguidade.

```
<!ENTITY booleano "(sim | não)" "não")>

<!-- elemento de nível superior, raíz. O elemento catálogo é uma lista de itens -- >
<!ELEMENT catálogo (item)+>
<!-- elemento item é composto por um nome, seguido por endereços, telefones, faxes e
emails -- >
<!ELEMENT item (nome, endereço*, telefone*, fax*, email*)>

<!-- elemento nome é composto por caracteres de dados - texto - ou primeiro nome ou
sobrenome -- >
<!ELEMENT nome (#PCDATA | pnome | snome)*>
<!ELEMENT pnome (#PCDATA)>
<!ELEMENT snome (#PCDATA)>

<!-- elementos endereço, telefone, fax e email possuem atributo 'preferido'-- >
<!ELEMENT endereço (rua, estado?, CEP, cidade, pais)>
<!ATTLIST endereço preferido &booleano;>
<!ELEMENT rua     (#PCDATA)>
<!ELEMENT estado (#PCDATA)>
<!ELEMENT CEP    (#PCDATA)>
<!ELEMENT cidade (#PCDATA)>
<!ELEMENT pais    (#PCDATA)>

<!ELEMENT telefone (#PCDATA)>
<!ATTLIST telefone preferido &booleano;>

<!ELEMENT fax (#PCDATA)>
<!ATTLIST fax preferido &booleano;>

<!ELEMENT email (#PCDATA)>
<!ATTLIST email preferido &booleano;
                    href CDATA #REQUIRED >
```

FIGURA 13 – EXEMPLO DE DOCUMENTO DTD

Neste capítulo foi apresentado o documento XML – sua sintaxe e marcadores – bem como o documento DTD – sua estrutura, seus construtores e a forma de definir seus elementos. No próximo capítulo será apresentado um algoritmo e o mapeamento entre um banco de dados integrado e seus bancos componentes, baseado em XML.

7 ALGORITMO E MAPEAMENTO DAS CONSULTAS

A decomposição de uma operação global em sub-operações é necessária para a manipulação de dados através de bancos de dados integrados, sua problemática é equivalente à do mapeamento de visões e as técnicas para alterações, ao longo da história, vem se deparando em inúmeros limites. Os dados de uma visão podem ser alterados somente se:

- A operação que gerou a visão não contiver funções de agregação, agrupamento, registros máximos ou mínimos, distinção ou união em sua cláusula;
- A operação que gerou a visão não possuir atributos derivados, ou seja, não pode possuir atributos que sejam resultado do conjunto de outros atributos;
- A operação que gerou a visão não possuir atributos que não sejam provenientes de tabelas, por exemplo variáveis do sistema, data corrente;
- A operação que gerou a visão ser composta somente por uma tabela.

Além da decomposição, utilizada também pelo controle de transação (Capítulo 5), a tradução das sub-operações para a linguagem de manipulação de dados do SGBD local é inevitável, pois os SGBDs locais podem possuir diferentes capacidades descritivas de suas operações. Por exemplo, SGBDs no modelo hierárquico ou de rede não suportam todas funções de agregação ou de agrupamento, que SGBDs no modelo relacional normalmente suportam.

Para que a decomposição destas operações seja possível, é necessário que haja um mapeamento entre as entidades do esquema integrado e as entidades dos esquemas locais componentes. Este mapeamento deve informar as características de

cada esquema local, como a estrutura e o tipo de dado de cada objeto, para que não haja conflito entre a passagem pelos diferentes bancos de dados envolvidos.

A proposta deste capítulo é um algoritmo que permite que operações apresentadas ao sistema integrado como um pacote único, sejam decompostas em sub-operações e traduzidas de acordo com as características dos bancos de dados locais envolvidos na integração, para que sejam submetidas a estes bancos locais. A Figura 14 apresenta a arquitetura básica deste modelo.

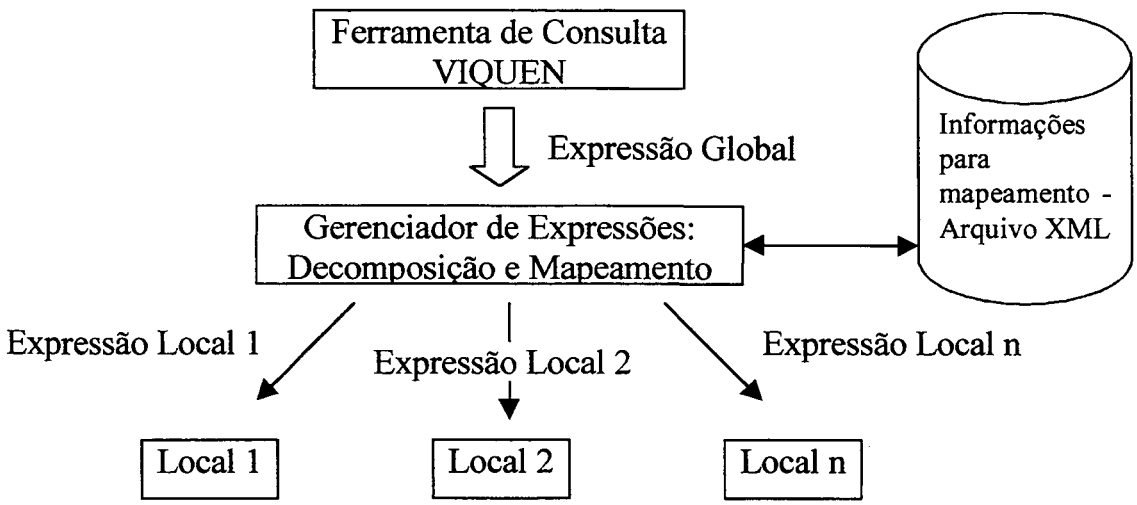


FIGURA 14 – ARQUITETURA DO PROTÓTIPO

A arquitetura deste protótipo está centrada no algoritmo de decomposição e mapeamento, este recebe uma expressão formulada sobre o esquema global e baseada na álgebra do modelo objeto/relacional, depois mapea esta expressão para cada um dos esquemas locais que compõem o esquema integrado. O algoritmo lê a expressão de entrada e, de acordo com as informações de cada esquema local armazenadas em um arquivo XML, gera as expressões locais.

A expressão global de entrada pode vir de qualquer ferramenta de consulta. Um exemplo de ferramenta que possui este pré-requisito, é o VIQUEN, um sistema de visualização de esquemas e de consulta visual que opera sobre esquemas relacionais e utiliza o modelo semântico objeto/relacional para interagir com o usuário.

O VIQUEN permite consulta a somente uma base de dados mapeando os operadores algébricos do modelo objeto/relacional para o SQL. Daí a importância do algoritmo, que possibilita a extensão desta ferramenta para trabalhar com uma base integrada e acessar várias outras bases de forma transparente.

Nas próximas subseções serão apresentados um modelo para este mapeamento baseado na linguagem padrão XML, apresentada no Capítulo 6, e um algoritmo para a decomposição das operações globais.

7.1 MAPEAMENTO

O mapeamento entre o esquema integrado e os esquemas locais subjacentes tipicamente mantém informações como correspondência de nomes, composição, grau de abstração, escalas e mapeamentos de valores.

Para uma completa troca de significados entre os esquemas e ser possível o acesso integrado, suas diferenças devem ser resolvidas. Para permitir a resolução destas diferenças de representação e/ou arquitetura, o mapeamento abrange:

- Correspondências entre propriedades semânticas equivalentes (como entre duas classes de diferentes esquemas, ou entre dois atributos diretos);
- Correspondência entre propriedades semânticas diferentes (como entre classes e atributos).

Este capítulo propõe um mapeamento para efetuar a correspondência entre objetos do esquema integrado e os objetos dos esquemas que compõem o esquema integrado, por meio de documentos XML e sua definição formal – o DTD – (apresentados no Capítulo 6).

Os documentos XML descrevem cada objeto do esquema integrado com base nos objetos originais que o compõem. Para cada objeto do esquema

integrado será apresentado sua regra de integração – disjunta, contem, interseção ou igual (apresentadas no Capítulo 4) –, cada uma de suas entidades, com informações dos seus bancos de dados de origem, sua localização, a descrição e a função de mapeamento de seus atributos.

A Figura 15 apresenta a especificação formal do documento XML, utilizando para isso o padrão DTD (vide Seção 6.2). Embora a W3C estar trabalhando em um padrão de maior expressividade para a especificação de um documento XML, conhecido como "*Schema*", este trabalho adotará o padrão o DTD. Por ele suportar de forma adequada os de tipos de dados, subclasses e modelos de conteúdo da linguagem SQL e, também, pelo padrão "*XML Schema*" não ter sido completamente finalizado até a data deste trabalho. O documento DTD defini a estrutura de cada um dos elementos do documento XML, como os atributos e a relação entre eles. A Figura 18 apresenta um fragmento do documento XML gerado sobre o esquema integrado apresentado no Exemplo 7.1

```
<! -- ARQUIVO DTD -- >

<!ELEMENT modelo (Objeto)+ >
<!ELEMENT Objeto (nome, regra, obj_componente*, atributo*)+>
<!ELEMENT nome (#PCDATA)>
<!ELEMENT regra (igual | contem | disjunta | interseção)>
<!ELEMENT obj_componente (#PCDATA)>
<!ATTLIST obj_componente banco_dados #REQUIRED>
<!ELEMENT Atributo (nome, atrib_componente*)+>
<!ELEMENT atrib_componente (nome, mapeamento*, atrib_identifica*)>
<!ATTLIST atrib_componente  objeto #REQUIRED
                            regra  #REQUIRED
                            tipo (atômico | tabela | multivalorado ) "atômico">
<!ELEMENT atrib_identifica (nome, mapeamento*)>
<!ATTLIST atrib_identifica     regra  #REQUIRED
<!ELEMENT mapeamento (função | valor ) >
<!ELEMENT função (#PCDATA) >
<!ELEMENT valor  (#PCDATA)>
<!ATTLIST valor  valor_integrado #PCDATA
                 valor_original #PCDATA
```

FIGURA 15 – ARQUIVO DTD: ESTRUTURA DO MAPEAMENTO PROPOSTO

No mapeamento proposto, o arquivo DTD apresenta o esquema integrado em relação aos esquemas originais. Segue a descrição de cada um de seus elementos e atributos.

Elemento *modelo*: representa o esquema integrado, composto por todas as suas classes de objetos.

Elemento *Objeto*: representa cada objeto do esquema integrado, quanto às suas características de integração e equivalência. Composto pelo nome do objeto no esquema integrado, pela regra de equivalência na qual foi gerado, pelos objetos originais que o compõem e pela descrição de seus atributos.

Elemento *Obj_componente*: lista os objetos que compõem um determinado objeto no esquema integrado, informando o nome do banco de dados de origem.

Elemento *Atributo*: representa cada atributo de um objeto no esquema integrado. Composto pelo seu nome e pelas informações dos atributos nos objetos originais que representa.

Elemento *Atrib_componente*: traz informações de cada atributo nos seus objetos originais, como nome, função de mapeamento, regra de equivalência e o tipo de atributo que representa. Tento como base para a regra de equivalência a sua relação para com o objeto integrado e não com os demais objetos, por exemplo, o atributo c do objeto AB no esquema integrado representa a interseção dos atributos a e b, pois a regra de equivalência entre eles é a β CONTEM b, porém a relação entre a e c é a β IGUAL c, pois DOM (c) = DOM (a), e a relação entre b e c é b β CONTIDO_EM c pois DOM (c) $\supset$ DOM (b).

Elemento *Mapeamento*: traz informações quanto ao mapeamento de valores entre o atributo integrado e o atributo original. Este mapeamento pode ser através de uma função (por exemplo, f(x) = 2x) ou através de relação direta (por exemplo, 1 = banana, 2 = maçã).

Elemento *Atrib_identifica*: Existem casos, principalmente nos atributos multivalorados, em que um atributo do esquema integrado corresponde a mais de um atributo nos esquemas componentes, podendo estes representar uma tabela, entre estes atributos está o identificador do objeto (ou da tabela). O elemento *Atrib_identifica* faz parte do elemento *Atrib_componente*.

Exemplo7.1: Seja *Pessoa* uma entidade do esquema integrado composto pela associação da entidade *Usuários_bib* do esquema BD01 e *Empregados* do esquema BD02. Pela análise dos bancos locais é possível deduzir que existem objetos em comum entre *Usuários_bib* e *Empregados*, ou seja, existem empregados que são usuários da biblioteca. Assim, a regra de equivalência entre eles é INTERSEÇÃO. A figura 16 mostra os esquemas locais, a Figura 17 a estrutura do modelo integrado e a Figura 18 o mapeamento destes, segundo modelo DTD apresentado.

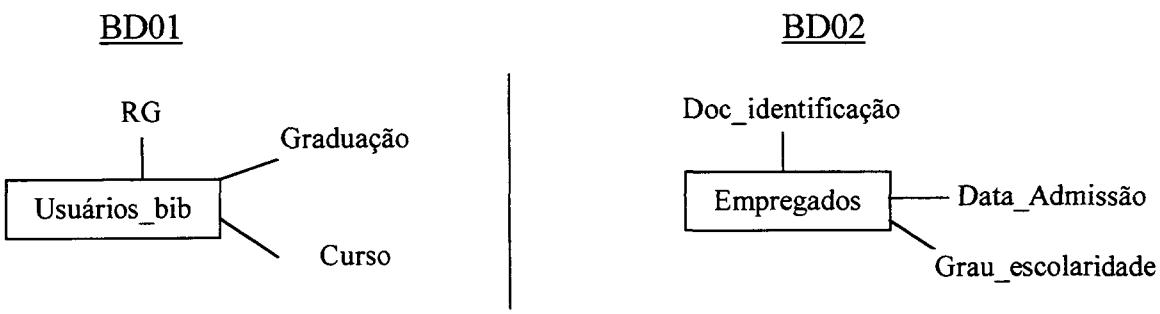


FIGURA 16 – DIAGRAMA DOS ESQUEMAS LOCAIS DO EXEMPLO 7.1

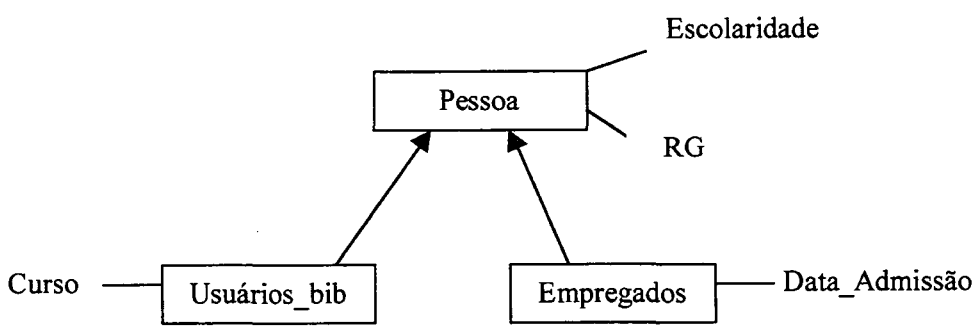


FIGURA 17 – DIAGRAMA DO ESQUEMA INTEGRADO DO EXEMPLO 7.1

```
<! -- ARQUIVO XML-- >

<?xml version="1.0"?>
<!DOCTYPE modelo SYSTEM MODELO.DTD>
<modelo>
  <Objeto>
    <nome> pessoa <\nome>
    <regra> interseção <\regra>
    <obj_componente banco_dados = "BD01"> Usuarios_bib <\obj_componente>
    <obj_componente banco_dados = "BD02"> Empregados <\obj_componente>
    <atributo>
       <nome> RG <\nome>
       <atrib_componente objeto = "Usuarios_bib" regra = "interseção">
          <nome> RG <\nome>
          <mapeamento>
                <função> f(x) = x <\função>
          <\mapeamento>
       <\atrib_componente>
       <atrib_componente objeto = "Empregados" regra = "interseção" >
           <nome> Doc_identificação <\nome>
           <mapeamento>
                <função> f(x) = x <\função>
           <\mapeamento>
       <\atrib_componente>
    <\atributo>
    <atributo>
        <nome> escolaridade <\nome>
        <atrib_componente objeto "Usuarios_bib" regra = "igual">
            <nome> graduação <\nome>
            <mapeamento>
                <função> f(x) = x <\função>
            <\mapeamento>
        <\atrib_componente>
        <atrib_componente objeto "Empregados" regra = "contem">
            <nome> grau_escolaridade <\nome>
            <mapeamento>
                <valor valor_integrado = "1" valor_original = "segundo grau"> <\valor>
                <valor valor_integrado = "2" valor_original = "terceiro grau"> <\valor>
                <valor valor_integrado = "3" valor_original = "pós-graduação"> <\valor>
                <valor valor_integrado = "4" valor_original = "mestrado">  <\valor>
                <valor valor_integrado = "4" valor_original = "doutorado"> <\valor>
            <\mapeamento>
        <\atrib_componente>
    <\atributo>
  <\Objeto>
  <Objeto>
    <nome> Usuários _Bib <\nome>
    <regra> igual <\regra>
    <obj_componente banco_dados = "DB01"> Usuarios_bib <\obj_componente>
    <atributo>
       <nome> curso <\nome>
       <atrib_componente objeto = "Usuarios_bib" regra = "igual">
          <nome> curso <\nome>
```

```
        <mapeamento>
              <função> f(x) = x <\função>
        <\mapeamento>
        <atrib_identifica regra = "igual">
           <nome> RG <\nome>
            <mapeamento>
              <função> f(x) =  x <\função>
            <\mapeamento>
         <\atrib_identifica>
  <\atributo>
 <\Objeto>
 <Objeto>
  <nome> Empregados <\nome>
  <regra> igual <\regra>
  <obj_componente banco_dados = "BD02"> Empregados <\obj_componente>
  <atributo>
     <nome> data_admissão <\nome>
     <atrib_componente objeto = "Empregados" regra = "igual">
        <nome> data_admissão <\nome>
        <mapeamento>
              <função> f(x) = x <\função>
        <\mapeamento>
        <atrib_identifica regra = "igual">
           <nome> Doc_identificação <\nome>
            <mapeamento>
              <função> f(x) =  x <\função>
            <\mapeamento>
         <\atrib_identifica>
  <\atributo>
 <\Objeto>
<\modelo>
```

FIGURA 18 – MAPEAMENTO EM XML DO ESQUEMA INTEGRADO DO EXEMPLO 7.1

O mapeamento proposto abrange também atributos complexos e multivalorados, como mostra o Exemplo 7.2.

Exemplo 7.2: Seja *Pessoa* um objeto do esquema integrado, cuja chave primária é representada pelo atributo *cod_ pessoa.* O objeto *Pessoa* é composto pela união dos objetos *Clientes*, banco de dados BD1, e *Funcionários*, banco de dados BD2. O objeto *Pessoa* possui um atributo *telefone* composto pelos atributos *celular, residencial* e *comercial*, onde *telefone* é a união dos atributos *celular, residencial* e *comercial* da tabela *Clientes* e *fone#1, fone#2* e *fone#3* da tabela *Funcionários*. E um atributo

nome_filhos multivalorado que representa a união da tabela *Cli_ filhos* de BD1 e do atributo multivalorado *nome_ filhação* da tabela *Funcionários*. A figura 19 mostra os esquemas locais, a Figura 20 a estrutura do modelo integrado e a Figura 21 o mapeamento destes, segundo modelo DTD apresentado.

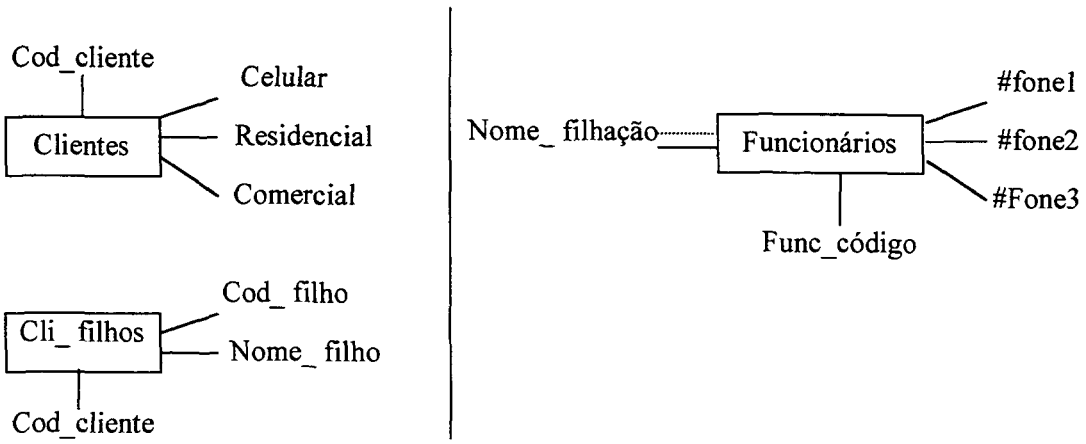


FIGURA 19 – DIAGRAMA DOS ESQUEMAS LOCAIS DO EXEMPLO 7.2

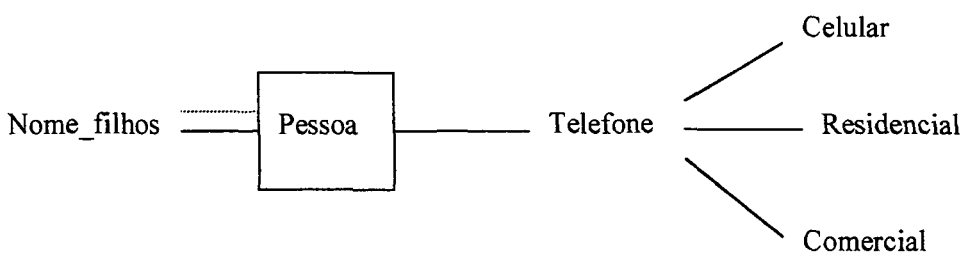


FIGURA 20 – DIAGRAMA DO ESQUEMA INTEGRADO DO EXEMPLO 7.2

```
<! -- ARQUIVO XML-- >

<?xml version="1.0"?>
<!DOCTYPE modelo SYSTEM MODELO.DTD>
<modelo>
  <Objeto>
    <nome> pessoa <\nome>
    <regra> interseção <\regra>
    <obj_componente banco_dados = "BD1"> Clientes <\obj_componente>
    <obj_componente banco_dados = "BD1"> Cli_filhos <\obj_componente>
    <obj_componente banco_dados = "BD2"> Funcionários <\obj_componente>
    <atributo>
       <nome> Cod_pessoa <\nome>
       <atrib_componente objeto = "Clientes" regra = "interseção">
          <nome> Cód_cliente <\nome>
           <mapeamento>
                <função> f(x) = x <\função>
```

```
        <\mapeamento>
    <\atrib_componente>
    <atrib_componente objeto = "Cli_filhos" regra = "interseção">
      <nome> Cód_cliente <\nome>
      <mapeamento>
          <função> f(x) = x <\função>
      <\mapeamento>
    <\atrib_componente>
    <atrib_componente objeto = "Funcionários" regra = "interseção" >
      <nome> Func_código <\nome>
    <\atrib_componente>
<\atributo>
<atributo>
    <nome> telefone.celular <\nome>
    <atrib_componente objeto "Clientes" regra = "interseção">
      <nome> celular <\nome>
      <mapeamento>
          <função> f(x) = x <\função>
      <\mapeamento>
    <\atrib_componente>
    <atrib_componente objeto "Funcionários" regra = "interseção">
      <nome> fone#1 <\nome>
      <mapeamento>
          <função> f(x) = x <\função>
      <\mapeamento>
    <\atrib_componente>
<\atributo>
<atributo>
    <nome> telefone.residencial <\nome>
    <atrib_componente objeto "Clientes" regra = "interseção">
      <nome> residencial <\nome>
      <mapeamento>
          <função> f(x) = x <\função>
      <\mapeamento>
    <\atrib_componente>
    <atrib_componente objeto "Funcionários" regra = "interseção">
      <nome> fone#2 <\nome>
      <mapeamento>
          <função> f(x) = x <\função>
      <\mapeamento>
    <\atrib_componente>
<\atributo>
<atributo>
    <nome> telefone.comercial<\nome>
    <atrib_componente objeto "Clientes" regra = "interseção">
      <nome> comercial <\nome>
      <mapeamento>
          <função> f(x) = x <\função>
      <\mapeamento>
    <\atrib_componente>
    <atrib_componente objeto "Funcionários" regra = "interseção">
      <nome> fone#3 <\nome>
      <mapeamento>
```

```
                <função> f(x) = x <\função>
            <\mapeamento>
        <\atrib_componente>
    <\atributo>
    <atributo>
        <nome> nome_filhos <\nome>
        <atrib_componente objeto "Cli_filhos" regra = "interseção" tipo = "tabela">
            <nome> nome_filhos <\nome>
            <mapeamento>
                <função> f(x) = x <\função>
            <\mapeamento>
            <atrib_identifica regra = "interseção">
                <nome> cod_filhos <\nome>
                <mapeamento>
                    <função> f(x) = CharToInt( x )<\função>
                <\mapeamento>
            <\atrib_identifica>
        <\atrib_componente>
        <atrib_componente objeto "Funcionários" regra = "interseção" tipo = "multivalorado">
            <nome> nome_filiação <\nome>
            <mapeamento>
                <função> f(x) = x <\função>
            <\mapeamento>
        <\atrib_componente>
    <\atributo>
  <\Objeto>
<\modelo>
```

FIGURA 21 – MAPEAMENTO DO EXEMPLO 7.2 SEGUNDO MODELO PROPOSTO

No exemplo acima o atributo nome_filhos possui o elemento *Atrib_identifica*, que corresponde a nome_filhos e a código_filhos (chave da tabela Cli_filhos, onde estão estes atributos no objeto componente).

7.2 ALGORITMO PARA DECOMPOSIÇÃO DE OPERAÇÕES GLOBAIS

Devido à heterogeneidade e à autonomia dos bancos de dados locais envolvidos, o algoritmo deve ser capaz de fornecer operações diferentes de acordo com o modelo e a plataforma de cada SGBD local. Como dito anteriormente, este trabalho apresenta tanto as operações globais quanto as sub-operações sempre em uma

mesma linguagem, no caso SQL, por ser uma linguagem de consulta padrão e utilizada pela grande maioria dos SGBDs encontrados, hoje, no mercado.

A manipulação em um banco de dados é feita através de inclusão, alteração e exclusão das instâncias de seus objetos. Quando esta manipulação é feita via esquema integrado, o mapeamento não informa com exatidão em qual das bases componentes a instância, que está sendo alterada, se encontra. Assim, nos casos de alteração, a regra é tentativa e erro, ou seja, aplica a manipulação desejada em todas as bases componentes, independente da regra de correspondência. Nos casos de inclusão e exclusão, é necessário informação adicional vinda do usuário, a menos que a regra de equivalência seja β IGUAL. Isto é, se o objeto AB do esquema integrado for composto pela união de A e B, e deseja-se incluir (ou excluir) uma nova instância em AB, como definir se esta nova instância deve ser incluída (ou excluída) em A, B ou em ambos. Por outro lado, se A e B forem exatamente iguais, deduz-se que a instância deva ser incluída (ou excluída) em ambos.

De acordo com as regras de equivalência (vide Capítulo 4) é permitido alterações nos valores do banco de dados integrado, isto é possível porque a função de mapeamento define uma correspondência entre os valores no esquema integrado e os valores nos bancos de dados originais, abrangendo todos os valores possíveis. Quando a ação é inclusão, o usuário deve informar valores do esquema integrado, desde que estes estejam mapeados, para então serem armazenados nos bancos de dados originais. Entretanto, no caso da inclusão, se o valor escolhido não possuir mapeamento para o banco de dados original, a ação não poderá ser concluída. Isto é, se o objeto AB do esquema integrado, composto pela união de A e B, possuir o atributo ab' composto por a e b, respectivamente, e a correspondência entre a e ab for ∀ab = 'xx', a = 'x1', ∀ab = 'yy', a = 'y1'. Ao incluir uma nova instância em AB, cujo valor de ab = 'rr' não encontra-se valor correspondente para a no mapeamento.

Se a relação de equivalência entre os atributos envolvidos for β IGUAL, então a capacidade de alteração, inclusão e exclusão é total, pois a função de

mapeamento e a sua inversa são totais em todos os domínios. Quando a função de mapeamento não abrange todos os valores dos domínios envolvidos, a capacidade de alteração não é total, ou seja, podem existir instâncias de objetos nas quais o mapeamento não foi definido. Isto ocorre nas situações de equivalência β CONTEM, β CONTIDO_EM e β INTERSERÇÃO.

Os algoritmos apresentados a seguir respeitam as estratégias de integração e as regras de equivalência entre objetos e atributos para efetuar a decomposição da operação no esquema integrado para as sub-operações nos seus bancos de dados de origem. Onde, Valor_Elemento(nome_elemento), Valor_Atributo(nome_atributo), Possui_Elemento(nome_elemento) e Procura_Elemento(nome_elemento, valor) são funções que devem utilizar os padrões criados pela W3C, como o Document Objet Model (vide Capítulo 6), para acessar os dados em uma estrutura XML. Os algoritmos mostram as expressões de entrada sempre com o operador "*and*[23]" a título de ilustração, porém isto não impede que elas possuam outros operadores da linguagem SQL, por exemplo o operador "*or*[24]".

Função Mapeamento
Entrada: Ponteiro para Atributo, Valor 'X' do atributo na expressão.
Saída: Valor 'X' Mapeado.

```
Se Possui_Elemento( mapeamento ) então
        Se Possui_Elemento( função ) então
                Retorna  Executa( Valor_Elemento( função), "xi" );
        Se Possui_Elemento( valor ) então
                Se Procura_Elemento( valor, "xi" ) = 1 então
                    Retorna Valor_Atributo( valor_original );
                Se Procura_Elemento( valor, "xi" ) = 0 então
                    Retorna 'Erro – Falta de mapeamento';
                Se Procura_Elemento( valor, "xi" ) > 1 então
                    Retorna 'Erro – Para um mesmo valor no dom. integrado existe
                            mais de um valor correspondente no dom. original';
Se não
        Se Valor_Elemento( regra ) == "igual" então
```

[23] Do inglês E.

[24] Do inglês OU.

```
                Retorna "xi";
        Se não
                Retorna 'Erro – Falta de mapeamento';
```

Algoritmo 1

Entrada: update E_1 set e_1 = 'x_1', ..., e_n = 'x_n' where e_i = 'x_i' and ... and e_k = 'x_k' , onde i, k □ 0
Saída: Vetor com expressões decompostas

```
Para cada obj_componente l de E1
        Banco_dados[l] := Valor_Atributo( database )
        Obj_Original[l] := Valor_Elemento( obj_componente );
        Para cada atributo en da cláusula SET
            Para cada atrib_componente de en
                Se Obj_Original[l] == Valor_Atributo ( objeto ) então
                    Atributo[l, n] := Valor_Elemento( nome );
                    Se Valor_Atributo( tipo ) == "tabela"
                            DecompValor( ValorA, X, n ):
                    Se não
                            ValorA[l, n] := Mapeamento( atrib_componente, 'X' );
                    Se Possui_Elemento( atrib_identifica ) então
                            Where[l, i] := Valor_Elemento( nome );
                            ValorW[l, i] := Mapeamento(atrib_identifica, 'X');
        Para cada atributo ei da cláusula WHERE
            Para cada atrib_componente de ei
                Se Obj_Original[l] == Valor_Atributo ( objeto ) então
                    Where[l, i] := Valor_Elemento( nome );
                    Se Valor_Atributo( tipo ) == "tabela"
                            DecompValor( ValorW, 'X', i );
                    Se não
                            ValorW[l, i] := Mapeamento( atrib_componente, 'X' );
                    Se Possui_Elemento( atrib_identifica ) então
                            Where[l, i] := Valor_Elemento( nome );
                            ValorW[l, i] := Mapeamento(atrib_identifica, 'X');
Para l de 0 a Total( Obj_Original )
        Para n de 0 a Total( Atributo[l, *] )
                Se Atributo[l, n] == "Erro"
                    Expressão[l] := "Erro";
                    Retornar Loop Externo;
                Expressão[l] := Expressão[l] + Atributo[l, n] + " = " + ValorA[l, n] + ",";
        Para i de 0 a Total( Where[l, *] )
                Se Where[l, i] == "Erro"
                    Expressão[l] := "Erro";
                    Retornar Loop Externo;
                Se i == 0 então
                    Expressão[l] := Expressão[l] + " Where" + Where[l, i] + " = " + ValorW[ l, i];
                Se não
                    Expressão[l] := Expressão[l] + " and " + Where[l, i] + " = " + ValorW[ l, i];
        Se Expressão[l] != "" então
                Expressão[l] := "Update " + Banco_Dados[l] + Obj_Original[l] + " Set " +
                                Expressão[l];'
Retorna Expressão
```

Algoritmo 2

Entrada: delete E_1 where e_i = 'x_i' and ... and e_k = 'x_k' , onde i, k □ 0
Saída: Vetor com expressões decompostas

```
Se Valor_Elemento( regra ) != "igual"
    Erro – regra de equivalência não permite exclusão
Se não
    Para cada obj_componente l de E1
        Banco_dados[l] := Valor_Atributo( database )
        Obj_Original[l] := Valor_Elemento( obj_componente );
        Para cada atributo ei da cláusula WHERE
            Para cada atrib_componente de ei
                Se Obj_Original[l] == Valor_Atributo ( objeto ) então
                    Where[l, i] := Valor_Elemento( nome );
                    Se Valor_Atributo( tipo ) == "tabela"
                            DecompValor( ValorW, 'X', i );
                    Se não
                            ValorW[l, i] := Mapeamento( atrib_componente, 'X' );
                    Se Possui_Elemento( atrib_identifica ) então
                            Where[l, i] := Valor_Elemento( nome );
                            ValorW[l, i] := Mapeamento(atrib_identifica, 'X');
Para l de 0 a Total( Obj_Original )
        Para i de 0 a Total( Where[l, *] )
                Se Where[i, n] == "Erro"
                    Expressão[l] := "Erro";
                    Retornar Loop Externo;
                Se i == 0 então
                    Expressão[l] := Expressão[l] + "Where" + Where[l, i] + " = " + ValorW[l, i];
                Se não
                    Expressão[l] := Expressão[l] + " and " + Where[l, i] + " = " + ValorW[l, i];
        Se Expressão[l] != "" então
                Expressão[l] := "Delete " + Banco_Dados[l] + Obj_Original[l] +
                                Expressão[l];
Retorna Expressão
```

Algoritmo 3

Entrada: insert into E_1 (e_1, ..., e_n) values ('x_1', ..., 'x_n')
Saída: Vetor com expressões decompostas

```
Se Valor_Elemento( regra ) !=  "igual"
    Erro – regra de equivalência não permite inclusão
Se não
    Para cada obj_componente l de E1
        Banco_dados[l] := Valor_Atributo( database )
        Obj_Original[l] := Valor_Elemento( obj_componente );
        Para cada atributo en da cláusula INSERT
            Identificar valor 'X' correspondente na cláusula VALUES
            Para cada atrib_componente de en
                Se Obj_Original[l] == Valor_Atributo ( objeto ) então
                    Atributo[l, n] := Valor_Elemento( nome );
                    Se Valor_Atributo( tipo ) == "tabela"
                            DecompValor( ValorA, 'X', n );
                    Se não
```

```
                                ValorA[l, n] := Mapeamento( atrib_componente, 'X' );
Para l de 0 a Total( Obj_Original )
        Para n de 0 a Total( Atributo[l, *] )
                Se Atributo[l, n] == "Erro"
                    Expressão[l] := "Erro";
                    Retornar Loop Externo;
                Expressão[l] := Expressão[l] + Atributo[l, n] + ",";
        Se Expressão[l] != "" então
                Expressão[l] := "Insert into" + Banco_Dados[l] + Obj_Original[l] + Expressão[l] +
                                " Values ";
        Para n de 0 a Total( ValorA[l, *] )
                Expressão[l] := Expressão[l] + ValorA[l, n] + ",";
Retorna Expressão
```

Nas próximas subseções será mostrado que os algoritmos propostos convertem a expressão principal em sub-expressões de acordo com as características dos esquemas envolvidos.

7.2.1 Quanto as Regras de Integração e Mapeamento

Exemplo 7.3: Neste exemplo serão mostrados os principais passos dos três algoritmos apresentados. Considerando os mapeamentos das Figuras 18 e 21.

1° Caso – Algoritmo1

Expressão de entrada: Update pessoa Set escolaridade = 2 Where RG = '123.456.789-90'

Banco_dados = {"BD01", "BD02"}
Obj_Original = {"Usuarios_bib", "Empregados"}
Atributo[0, *] = {"graduação"} ValorA[0, *] = {"2"}
Atributo[1, *] = {"grau_escolaridade "} ValorA[1, *] = {"terceiro grau"}
Where[0, *] = { "RG "} ValorW[0, *] = {" '123.456.789-90' "}
Where[1, *] = { "Doc_identificacao"} ValorW[1, *] = {" '123.456.789-90' "}
Expressao = { "Update BD01.Usuario_bib set graduação = 2 where RG = '123.456.789-90' ", "Update BD02.Empregados set grau_escolaridade = 'terceiro grau' where Doc_identificacao = '123.456.789-90' }

2° Caso – Algoritmo1

Expressão de entrada: Update pessoa Set escolaridade = 5 Where RG = '123.456.789-90'

Banco_dados = {"BD01", "BD02"}
Obj_Original = {"Usuarios_bib", "Empregados"}
Atributo[0, *] = {"graduação"} ValorA[0, *] = {"5"}
Atributo[1, *] = {Erro – Falta de mapeamento}
Where[0, *] = { "RG "} ValorW[0, *] = {" '123.456.789-90' "}
Where[1, *] = { "Doc_identificacao"} ValorW[1, *] = {" '123.456.789-90' "}
Expressao = { "Update BD01.Usuario_bib set graduação = 2 where RG = '123.456.789-90' ", "ERRO"}

3° Caso – Algoritmo2

Expressão de entrada: Delete from pessoa Where RG = '123.456.789-90'

Erro – Regra de equivalência não permite exclusão

4° Caso – Algoritmo3

Expressão de entrada: Insert into pessoa (RG, escolaridade) values (123.456.789-99, 3)

Erro – Regra de equivalência não permite inclusão

5° Caso – Algoritmo2

Expressão de entrada: Delete from Usuários_bib Where RG = '123.456.789-90'

Banco_dados = {"BD01"}
Obj_Original = {"Usuarios_bib"}
Where[0, *] = { "RG "} ValorW[0, *] = {" '123.456.789-90' "}
Expressao = { "Delete from BD01.Usuario_bib where RG = '123.456.789-90' "}

6° Caso – Algoritmo3

Expressão de entrada: Insert into Empregados (RG, Data_admissão) values ('123.456.789-90', '01/02/2002')

Banco_dados = { "BD02"}
Obj_Original = {"Empregados"}
Atributo[0, *] = {"Doc_identificação, Data_admissão"}
ValorA[0, *] = { '123.456.789-90', '01/02/2002'}
Expressao = : {"Insert into Empregados (Doc_identificação, Data_admissão) values ('123.456.789-90', '01/02/2002') "}

No 2° Caso, houve um erro por falta de mapeamento do objeto componente *Empregado* no atributo *escolaridade*, pois não existe mapeamento da escolaridade 5 no esquema integrado para o esquema componente BD02. Os algoritmos (o algoritmo 2 e o 3 possuem o mesmo comportamento) retornam as demais sub-operações (sem erro) e a indicação de erro, cabe ao administrador do banco de dados resolver se as executa ou não. Ou seja, a regra de executar somente se não houve decomposição com erro não cabe ao algoritmo, mas sim a uma análise da situação. No exemplo, escolaridade 5 poderia corresponder a pessoas com PhD e não existir nenhum empregado na instituição com tal titulação, porém existir usuários da biblioteca com PhD e a alteração ser desejada.

No 3° e 4° Caso, como a regra de equivalência é Usuários_bib ρ INTERSERÇÃO Empregados, a entidade *Pessoa* no esquema integrado é formada

pela união das duas entidades, ou seja, RWS(pessoa) = RWS(Usuários_bib) U RWS(Empregados). Neste caso, não há como saber se a instância excluída/inserida de *Pessoa* deve ser excluída/inserida da entidade *Usuários_Bib*, *Empregados* ou de ambas.

Como já dito anteriormente, nestes casos é necessário a interferência do usuário e/ou do administrador do banco de dados. Uma sugestão seria a inclusão de um atributo no mapeamento do objeto componente indicando se deve ou não sofrer inclusão e exclusão, porém esta é uma regra fixa que faz com que o esquema integrado perca características semânticas. Por exemplo, se ficar definido que sempre que houver inclusão de uma nova instância ela será incluída em *Usuários_bib*, desta forma nunca poderá ser incluído um novo empregado, ou se *Empregados* também possuir indicação de inclusão, toda nova instância incluída será obrigatoriamente usuário da biblioteca e empregado.

Uma outra sugestão é o usuário informar em tempo de execução aonde o registro de ser incluído/excluído. Neste caso os algoritmos deveriam possuir uma janela para entrada de dados, onde após lidos os objetos componentes seria questionado em quais deles executar a ação. Desta forma, seguindo o exemplo, poderia ser incluído/excluído tanto empregados quando usuários da biblioteca, não havendo perdas no esquema integrado, nem sintáticas nem semânticas.

Porém, a melhor solução para este exemplo é a inclusão diretamente no objeto desejado, como mostram o 5° e 6° caso.

7.2.2 Quanto a Atributos Compostos e Multivalorados

Exemplo 7.4: Neste exemplo será mostrado que os algoritmos apresentados suportam atributos compostos e multivalorados, considerando o mapeamento do Exemplo 7.2, apresentado na seção anterior.

1° Caso – Algoritmo1

Expressão de entrada: Update pessoa Set telefone.residencial = '222 6600', nome_filhos[\João] = 'João de Albuquerque' Where Cod_pessoa = 00125

Banco_dados = {"BD1", "BD1", "BD2"}
Obj_Original = {"Clientes", "Cli_filhos", "Funcionário"}
Atributo[0, *] = {"residencial "} ValorA[0, *] = {" '222 6600' "}
Atributo[1, *] = {"nome_filhos"} ValorA[1, *] = {" 'João de Albuquerque' "}
Atributo[2, *] = {"fone#2", nome_filiação[\João]} ValorA[2, *] = {" '222 6600', 'João de Albuquerque' "}
Where[0, *] = {"Cod_cliente"} ValorW[0, *] = {"00125"}
Where[1, *] = {"Cod_cliente", "cód_filho"} ValorW[1, *] = {"00125", "2"}
Where[2, *] = {"Func_Código "} ValorW[2, *] = {"00125"}
Expressao = {"Update BD1.Clientes Set residencial = '222 6600' where cod_cliente = 00125", "Update BD1.Cli_filhos Set nome_filhos = 'João de Albuquerque' where cod_cliente = 00125 and cód_filho = 2 ", "Update BD2.Funcionários Set fone#2 = 222 6600, nome_filiação[\João] = 'João de Albuquerque' Where func_código = 00125" }

2° Caso – Algoritmo2

Expressão de entrada: Delete from pessoa Where cod_pessoa = 00225

Erro – Regra de equivalência não permite exclusão

3° Caso – Algoritmo3

Expressão de entrada: Insert into pessoa (cod_pessoa, telefone.celular, nomes_filhos) values (00556, 9901 0205, ('José da Silva', 'Maria da Silva'))

Erro – Regra de equivalência não permite inclusão

No 2° e 3° caso, ocorreu erro devido a regra de equivalência entre as classes objeto ser INTERSEÇÃO, porém como os algoritmos possuem a mesma estrutura e seguem a mesma lógica de construção das sub-expressões, os algoritmos 2 e 3 também suportam atributos compostos e multivalorados.

7.2.3 Quanto as Estratégias de Integração

Seja A e B duas classes objeto que compõem o objeto AB no esquema integrado, e seja a e b seus atributos, respectivamente, que compõem o atributo ab. Seja A' uma subclasse de AB composta por A e a' seu atributo e seja B' uma subclasse de AB composta por B e b' seu atributo.

A Figura 22 mostra o mapeamento de AB, A' e B' em relação a A e B. Em seguida, para cada estratégia de integração entre A e B, apresentadas nas Figuras 1 a 4, será mostrado que os algoritmos propostos podem ser utilizados em todas as situações.

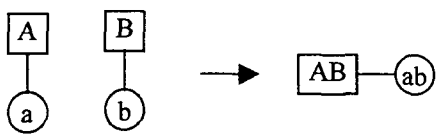
```
<! – ARQUIVO XML-- >

<?xml version="1.0"?>
<!DOCTYPE modelo SYSTEM MODELO.DTD>
<modelo>
  <Objeto>
    <nome> AB <\nome>
    <regra> XXXX <\regra>          <!—Definida em cada Caso →
    <obj_componente banco_dados = "BD1"> A <\obj_componente>
    <obj_componente banco_dados = "BD2"> B <\obj_componente>
    <atributo>
       <nome> ab <\nome>
       <atrib_componente objeto = "A" regra = XXXX>
          <nome> a <\nome>
          <mapeamento>
                <função> f(x) = x <\função>
          <\mapeamento>
       <\atrib_componente>
       <atrib_componente objeto = "B" regra = XXXX>
          <nome> b <\nome>
          <mapeamento>
                <função> f(x) = x <\função>
          <\mapeamento>
       <\atrib_componente>
    <\atributo>
  <\Objeto>
  <Objeto>
    <nome> A' <\nome>
    <regra> igual <\regra>
    <obj_componente banco_dados = "BD1"> A <\obj_componente>
    <atributo>
       <nome> a' <\nome>
       <atrib_componente objeto = "A" regra = "igual" >
          <nome> a <\nome>
       <\atrib_componente>
    <\atributo>
  <\Objeto>
  <Objeto>
    <nome> B' <\nome>
    <regra> igual <\regra>
    <obj_componente banco_dados = "BD2"> B <\obj_componente>
    <atributo>
       <nome> b' <\nome>
       <atrib_componente objeto = "B" regra = "igual" >
```

```
        <nome> b <\nome>
      <\atrib_componente>
    <\atributo>
  <\Objeto>
<\modelo>
```

FIGURA 22 – MAPEAMENTO PADRÃO DE AB

1° Caso: Regra A ρ IGUAL B

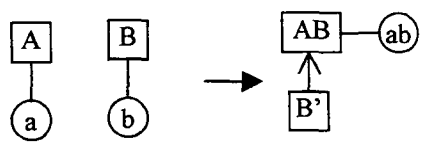
1) a β IGUAL b



Expressão de entrada1: Update AB Set ab = 'x' Where ab = 'y'
Expressão de entrada2: Delete from AB Where ab = 'y'
Expressão de entrada3: Insert into AB (ab) Values ('x')
Banco_dados = {"BD1", "BD2"}
Obj_Original = {"A", "B"}
Atributo[0, *] = {"a"} ValorA[0, *] = {"x"}
Atributo[1, *] = {"b"} ValorA[1, *] = {"x"}
Where[0, *] = {"a"} ValorW[0, *] = {"y"}
Where[2, *] = {"b"} ValorW[1, *] = {"y"}
Expressão1 = { "Update BD1.A Set a = 'x' where a = 'y'", "Update BD2.B Set b = 'x' where b = 'y' }
Expressão2 = { "Delete from BD1.A Where a = 'y'", "Delete from BD2.B Where b = 'y'"}
Expressão3 = { "Insert into BD1.A (a) Values ('x')", "Insert into BD2.B (b) Values ('x')" }

2° Caso: Regra A ρ CONTEM B (ou A ρ CONTIDO_EM B)

1) a β IGUAL b ou a β CONTEM b (ou a β CONTIDO_EM b)



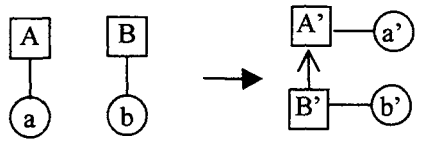
RWS(A) = RWS(AB)

DOM(ab) = DOM(a)

Expressão de entrada1: Update AB Set ab = 'x' Where ab = 'y'
Expressão de entrada2: Delete from AB Where ab = 'y'
Expressão de entrada3: Insert into AB (ab) Values ('x')
Banco_dados = {"BD1", "BD2"}
Obj_Original = {"A", "B"}
Atributo[0, *] = {"a"} ValorA[0, *] = {"x"}
Atributo[1, *] = {"b"} ValorA[1, *] = {"x"}
Where[0, *] = {"a"} ValorW[0, *] = {"y"}
Where[2, *] = {"b"} ValorW[1, *] = {"y"}
Expressão1 = { "Update BD1.A Set a = 'x' where a = 'y'", "Update BD2.B Set b = 'x' where b = 'y' }

Expressão2 = {"Erro – Regra de equivalência não permite exclusão"}
Expressão3 = {"Erro – Regra de equivalência não permite inclusão" }

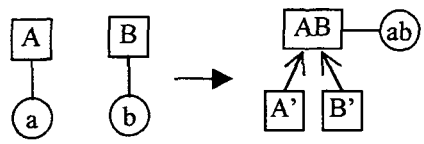
2) a β CONTEM b (ou a β CONTIDO_EM b)



Expressão de entrada1: Update A' Set a' = 'x' Where ab = 'y'
Expressão de entrada2: Delete from A' Where a' = 'y'
Expressão de entrada3: Insert into A' (a') Values ('x')
Banco_dados = {"BD1"}
Obj_Original = {"A"}
Atributo[0, *] = {"a"} ValorA[0, *] = {"x"}
Where[0, *] = {"a"} ValorW[0, *] = {"y"}
Expressão1 = { "Update BD1.A Set a = 'x' where a = 'y'" }
Expressão2 = { "Delete from BD1.A Where a = 'y'" }
Expressão3 = { "Insert into BD1.A (a) Values ('x')" }

3° Caso: Regra interseção, A ρ INTERSEÇÃO B

1) a β IGUAL b ou a β CONTEM b (ou a β CONTIDO_EM b)

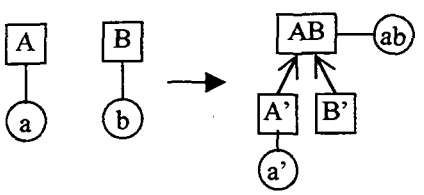


DOM(ab) = DOM(a) U DOM(b)

DOM(ab) = DOM(a) (ou DOM(ab) = DOM(b))

Expressão de entrada1: Update AB Set ab = 'x' Where ab = 'y'
Expressão de entrada2: Delete from AB Where ab = 'y'
Expressão de entrada3: Insert into AB (ab) Values ('x')
Banco_dados = {"BD1", "BD2"}
Obj_Original = {"A", "B"}
Atributo[0, *] = {"a"} ValorA[0, *] = {"x"}
Atributo[1, *] = {"b"} ValorA[1, *] = {"x"}
Where[0, *] = {"a"} ValorW[0, *] = {"y"}
Where[2, *] = {"b"} ValorW[1, *] = {"y"}
Expressão1 = { "Update BD1.A Set a = 'x' where a = 'y'", "Update BD2.B Set b = 'x' where b = 'y' }
Expressão2 = {"Erro – Regra de equivalência não permite exclusão"}
Expressão3 = { "Erro – Regra de equivalência não permite inclusão" }

2) a β CONTEM b (ou a β CONTIDO_EM b)



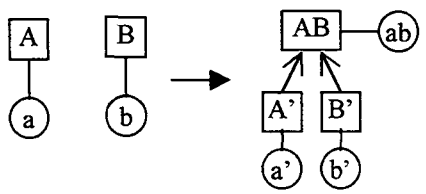
DOM(ab) = DOM(a) □ DOM(b)

DOM(ab) = DOM(b) (ou DOM(ab) = DOM(a))

Expressão de entrada1: Update AB Set ab = 'x' Where ab = 'y'
Expressão de entrada2: Delete from AB Where ab = 'y'

Expressão de entrada3: Insert into AB (ab) Values ('x')
Banco_dados = {"BD1", "BD2"}
Obj_Original = {"A", "B"}
Atributo[0, *] = {"a"} ValorA[0, *] = {"x"}
Atributo[1, *] = {"b"} ValorA[1, *] = {"x"}
Where[0, *] = {"a"} ValorW[0, *] = {"y"}
Where[2, *] = {"b"} ValorW[1, *] = {"y"}
Expressão1 = { "Update BD1.A Set a = 'x' where a = 'y'", "Update BD2.B Set b = 'x' where b = 'y' }
Expressão2 = {"Erro – Regra de equivalência não permite exclusão"}
Expressão3 = { "Erro – Regra de equivalência não permite inclusão" }

3) a β INTERSEÇÃO b



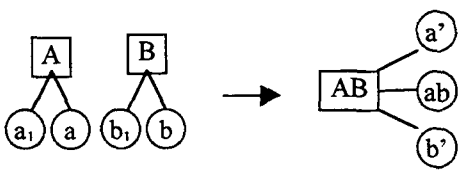
Expressão de entrada1: Update AB Set ab = 'x' Where ab = 'y'
Expressão de entrada2: Delete from AB Where ab = 'y'
Expressão de entrada3: Insert into AB (ab) Values ('x')
Banco_dados = {"BD1", "BD2"}
Obj_Original = {"A", "B"}
Atributo[0, *] = {"a"} ValorA[0, *] = {"x"}
Atributo[1, *] = {"b"} ValorA[1, *] = {"x"}
Where[0, *] = {"a"} ValorW[0, *] = {"y"}
Where[2, *] = {"b"} ValorW[1, *] = {"y"}
Expressão1 = { "Update BD1.A Set a = 'x' where a = 'y'", "Update BD2.B Set b = 'x' where b = 'y' }
Expressão2 = {"Erro – Regra de equivalência não permite exclusão"}
Expressão3 = { "Erro – Regra de equivalência não permite inclusão" }

7.2.4 Quanto a Integrações Livres

Seja A e B duas classes objeto que compõem o objeto AB no esquema integrado, seja a, a_1 atributos de A e b, b_1 atributos de B. O atributo ab é composto por a e b. Os atributos a_1 e b_1 não são relacionados. A Figura 25 mostra o mapeamento de AB em relação a A e B. Em seguida, será mostrado como se comportam os algoritmos propostos nesta situação.

```
<! -- ARQUIVO XML-- >

<?xml version="1.0"?>
<!DOCTYPE modelo SYSTEM MODELO.DTD>
<modelo>
  <Objeto>
    <nome> AB <\nome>
    <regra> igual  <\regra>
    <obj_componente banco_dados = "BD1"> A <\obj_componente>
    <obj_componente banco_dados = "BD2"> B <\obj_componente>
    <atributo>
       <nome> ab <\nome>
       <atrib_componente objeto = "A" regra = "igual" >
          <nome> a <\nome>
           <mapeamento>
                <função> f(x) = x <\função>
           <\mapeamento>
       <\atrib_componente>
       <atrib_componente objeto = "B" regra = "igual" >
          <nome> b <\nome>
           <mapeamento>
                <função> f(x) = x <\função>
           <\mapeamento>
       <\atrib_componente>
    <\atributo>
    <atributo>
       <nome> a' <\nome>
       <atrib_componente objeto = "A" regra = "igual" >
          <nome> a1 <\nome>
       <\atrib_componente>
    <\atributo>
    <atributo>
       <nome> b' <\nome>
       <atrib_componente objeto = "B" regra = "igual" >
          <nome> b1 <\nome>
       <\atrib_componente>
    <\atributo>
<\Objeto>
<\modelo>
```

FIGURA 23 – MAPEAMENTO DE AB



Expressão de entrada1: Update AB Set a' = 'x', b' = 'z' Where ab = 'y'
Expressão de entrada2: Delete from AB Where ab = 'y'
Expressão de entrada3: Insert into AB (ab, a', b') Values ('y', 'x', 'z')
Banco_dados = {"BD1", "BD2"}
Obj_Original = {"A", "B"}
Atributo[0, *] = {"a_1"} ValorA[0, *] = {"x"}

Atributo[1, *] = {"b_1"} ValorA[1, *] = {"z"}
Where[0, *] = {"a"} ValorW[0, *] = {"y"}
Where[2, *] = {"b"} ValorW[1, *] = {"y"}
Expressão1 = { "Update BD1.A Set a_1 = 'x' where a = 'y'", "Update BD2.B Set b_1 = 'z' where b = 'y' }
Expressão2 = { "Delete from BD1.A Where a = 'y'", "Delete from BD2.B Where b = 'y'"}
Expressão3 = { "Insert into BD1.A (a, a_1) Values ('y', 'x')", "Insert into BD1.B (b, b_1) Values ('y', 'z')" }

Este capítulo apresentou um algoritmo para a decomposição de operações, apresentadas ao sistema integrado como um pacote único, em sub-operações e traduzidas de acordo com as características dos bancos de dados locais envolvidos na integração, armazenadas em um arquivo XML. Mostrou também, que o algoritmo apresentado respeita as regras de integração e abrange os conceitos do modelo objeto/relacional.

8 CONCLUSÃO

Com a abertura de mercados e a globalização mundial, uma empresa precisa cada vez mais informações para manter-se ativa. Porém, nem sempre estas informações encontram-se centralizadas. Daí a necessidade de troca de conteúdo entre diferentes bases de dados. Uma solução muito utilizada, nestas situações, são os bancos de dados integrados. Uma das vantagens de utilizar um banco de dados integrado é a facilidade, para o usuário, de obtenção de informações provenientes de vários outros bancos de dados espalhados pelas organizações.

A contribuição deste trabalho foi mostrar que é possível, usando XML para descrever informações, obter uma estrutura padronizada e independente de plataforma, para interoperabilidade entre os esquemas integrados e seus esquemas componentes de forma transparente, sem a necessidade de reestruturação dos esquemas locais.

Para obter esta interoperabilidade, o trabalho apresenta algoritmos para a decomposição de uma expressão SQL, apresentada ao esquema integrado por intermédio de um sistema de visualização de esquemas, em expressões SQL correspondentes a cada esquema local.

As informações necessárias ao algoritmo, para que o banco de dados integrado interaja com os bancos de dados locais, estão estruturadas na forma um mapeamento escrito na linguagem XML. Estas informações são a base para a decomposição das expressões.

A solução apresentada permite SGBDs tanto no modelo relacional quanto no modelo objeto/relacional, desde que utilizem a linguagem SQL ou equivalente, assim engloba a maioria dos SGBDs atuais.

Como dito anteriormente (ver Capítulo 6) a XML é uma linguagem padronizada e possui um formato de dados auto descritivo, baseado em texto e universal. Devido a grande versatilidade e poder de descrição de dados da XML, a

estrutura dos modelos relacional e objeto/relacional pode ser precisamente descrita através de seus documentos de definição, os DTDs.

Com esta abordagem foi possível apresentar um modelo flexível e independente de plataforma. Pois, um documento XML carrega consigo tanto sua estrutura quanto suas informações e pode ser escrito utilizando vocabulário específico para cada domínio.

Os objetivos propostos foram alcançados da seguinte forma:

- Mapeamento do banco de dados integrado em relação aos bancos de dados locais que o compõem. Para isto foi aproveitada a flexibilidade e o poder de expressão da linguagem XML e do documento padrão que descreve sua estrutura, o DTD. Nestes documentos foram descritos todos os objetos do esquema integrado, os objetos locais que o compõem, as formas como se relacionam, conhecidas como regras de integração, seus atributos e valores correspondentes;
- Algoritmos para decomposição da expressão SQL apresentada ao esquema integrado, em expressões SQL correspondentes a cada esquema local. Apesar dos esquemas locais terem liberdade de estrutura, ou seja, não necessariamente falarem SQL, foi convencionado neste trabalho que as expressões locais seriam nesta linguagem. Esta característica limita sua implementação e restringe o uso a SGBDs cuja linguagem padrão seja SQL. Estes algoritmos estão baseados unicamente na estrutura do mapeamento, ou seja, no arquivo XML;
- Mapeamento e Algoritmo de decomposição preparados para a estrutura objeto/relacional. Tanto mapeamento quanto algoritmo suportam os conceitos de classe e herança e os conceitos de atributos multivalorados e compostos.

8.1 TRABALHOS FUTUROS

Entre as melhorias e os trabalhos que poderão utilizar como base o algoritmo e o mapeamento propostos, estão:

- Implementação do algoritmo;
- Adição de protocolos de controle de transação e de concorrência ao algoritmo, ao invés de somente retornar as expressões decompostas;
- Desenvolvimento de uma ferramenta que gere o esquema integrado e, a partir da forma como os objetos forem relacionados, gere automaticamente o arquivo DTD e o arquivo XML;
- Utilização o novo padrão XML "*Schema*" ao invés de DTD para definir a estrutura do arquivo XML.

REFERÊNCIAS BIBLIOGRÁFICAS

Ahmed, R.; et. Al. **The pegasus Heterogeneous Multidatabase System**. Computer society. Vol.24, nº 12, Outubro, 1998, p. 19-27.

Andersson, M.; Dupont, Y.; Tresch, M.; Ye, H. **FEMUS: A Federated Multilingual Database System.** Advanced Database Systems, Capítulo 18, 1993, p. 359-380. Disponível em <http://citeseer.nj.nec.com/andersson93femus.html>

Batini, C.; Lenzerini M. **A Comparative Analysis of Methodologies for Database Schema Integration**. ACM Computing Surveys, Vol. 18, n.º 4, Dezembro 1986.

Breitbart, Y.; Garcia-Molina, H.; Silberschatz, A. **Overview of multidatabase transaction management**. VLDB Journal, Vol. 1, n.º 2, 1992, p. 181-293. Disponível em: <http://citeseer.nj.nec.com/breitbart92overview.html>

Bosak, Jon. **Media-independent publishing: four myths about XML**. IEEE Computer society. Vol.31, nº 10, Outubro, 1998.

Bouguettaya, A.; Benatallah, B.; Elmagarmid, A. **An Overview of Multidatabase Systems: Past and Present**. Management of Heterogeneous and Autonomous Databases Systems, Morgan Kaufmann Publishers, 1999, Capítulo 1.

Buneman, P.; Davidson, S.; Kosky, A. **Theoretical Aspects of Schema Merging**. Proc. EDBT Conference, 1992. Disponível em: <http://citeseer.nj.nec.com/buneman92theoretical.html>

Chen, J.; Huang, Q.; Sajeev, A.S.M. **Interoperability between object-oriented programming languages and relational systems**. In Proceedings of TOOLS Conference, Melbourne, Dezembro 1995, p. 53-66, Prentice-Hall. Disponível em: <http://citeseer.nj.nec.com/2298.html>

Concurrency Control Service. CORBAservices, Concurrency Control, Vol. 1, Março 1995.

Date, C. J. **Introdução a Sistemas de Banco de dados**. 7.Ed. Rio de Janeiro: Campus, 2000.

Elmagarmid, A.; Du, W.; Ahmed, R **Local Autonomy and Its Effects on Multidatabase Systems**. Management of Heterogeneous and Autonomous Databases Systems, Morgan Kaufmann Publishers, 1999, Capítulo 2.

Elmasri, Ramez; Navathe, Shamkant B. **Fundamentals of Database Systems**. 2.Ed. Addison-Wesley Publishing Company, 1994.

Extensible Markup Language. World Wide Web Consortium (W3C). Disponível em: <http://www.w3c.org/XML>.

Gardarin G.; Gannouni, S.; Finance, B. **IRO-DB : A Distributed System Federating Object and Relational Databases.** Object-Oriented Multibase Systems, Prentice Hall, 1995. Disponível em: <http://citeseer.nj.nec.com/gardarin95irodb.html>

Georgakopoulos, D. **Multidatabase recoverability and recovery**. First International Workshop on Interoperability in Multidatabase Systems, 1991, p. 348-355. Disponível em: <http://citeseer.nj.nec.com/georgakopoulos91multidatabase.html>

Gogolla, Martin. **On Formal Semantics of Some Semantic Data Models**. Disponível em: <http://citeseer.nj.nec.com/132162.html>

Gueiber, Ezequiel. **VIQUEN – Um Ambiente Interativo para Consulta Visual e Extração de Esquemas**. Dissertação de mestrado. Departamento de Informática, Universidade Federal do Paraná, Curitiba, 2001.

Huang, X.; Miller, J. A. **Building a Web-Based Federated Simulation System with Jini and XML.** Disponível em: <citeseer.nj.nec.com/400714.html>

Jung, I.; Lee, J.; Moon S. **Concurrency Control in Multidatabase Systems: A Performance Study**. Journal of Systems Architecture, vol. 45, 1998, p. 97-114.

Kosky, A. S. **A Formal Model for Databases with Applications to Schema Merging**. Disponível em: <http://citeseer.nj.nec.com/94920.html>

Keim, D.; Kriegel, H.P.; Miethsam, A. **ObjectOriented Querying of Existing Relational Databases**. Disponível em: <http://citeseer.nj.nec.com/keim93objectoriented.html>

Landers, T.; Rosenberg R. **An Overview of Multibase**. H.J. Schneider, ed., Distributed Data Bases, 1982, p. 311-336.

Larson, J. A.; Navathe, S. B., Elmasri, R. **A Theory of Atribute Equivalence in Databases with Application to Schema Integration**. IEEE Transactions on Software Engineering, vol. 15, n.° 4, Abril 1989.

Marchal, Benoit. **XML – Conceitos e Aplicações**. São Paulo: Berkeley Brasil, 2000.

Martin, Didier et. Al. **Professional XML**. Wrox Press, 2000.

Nakano, Ryohei. **Translation with Optimazation from Relational Calculus to Relational Algebra Having Aggregate Functions**. ACM Transactions on Database Systems, Vol. 15, n.° 4, Dezembro 1990, p. 518-557.

Normas Brasileiras de Padronização. **Linguagens de Banco de dados – SQL**. Documento: NBR ISO/IEC 9075, 2000.

Oliveira, José P.M.; Edelweiss, Nina; Ribeiro, Cora P. **Sistema Distribuído para Intercâmbio de Dados de Saúde – SIDI**. Informática Pública. Vol. 3, n° 2, Dezembro, 2001.

Paredaens, J.; Gucht, D.V. **Converting Nested Algebra Expressions into Flat Algebra Expressions**. ACM Transactions on Database Systems. Vol. 17, n° 1, Março 1992, p. 65-93.

Pinto, José S.P. **X-VIQUEN – Uma Ferramenta Visual para Extração de Esquemas Utilizando XML**. Dissertação de mestrado. Departamento de Informática, Universidade Federal do Paraná, Curitiba, 2001.

Ram, S.; Ramesh, V. **Schema Integration: Past, Present and Future**. Management of Heterogeneous and Autonomous Databases Systems, Morgan Kaufmann Publishers, 1999, Capítulo 5.

Ramanathan, C. **Providing Object-Oriented Access to Existing Relational Databases**. Dissertação apresentada ao Departamento de Ciências da Computação, Universidade do Estado de Mississipi, Estados Unidos da América, Mississipi, 1997.

Scholl, M. H.; Laasch, C.; Tresch, M. **Updatable Views in Object-Oriented Databases**. Proc. of the 2 nd Int. Conf., DOOD '91. Alemanha, Munich: Springer, 1991. Disponível em: <http://citeseer.nj.nec.com/scholl91updatable.html>

Scopim, K. S.; Katsuragawa, R. **An Implementation of Database Integration Strategies**. Dissertação. Departamento de Informática, Universidade Federal do Paraná, 2001.

Silberschatz, A.; Korth, H.F. **Sistemas de Banco de dados**. São Paulo: Mc Graw-Hill, 1992.

Spaccapietra S.; Parent C. View Integration: **A Step Forward in Solving Structural Conflicts**. IEEE Transactions on knowledge and Data Engineering, Vol. 6, n.° 2, Abril 1994.

Spaccapietra S. et. Al. **ERC+: An Object+Relationship Paradigm for Database Applications**. Readings in Object-Oriented Systems, D. Rine(Ed.), IEEE Press, 1995.

Straube, D.D.; Ozsu, M.T. **Queries and Query processing in Object-oriented Database Systems**. ACM Transactions on Information Systems, vol. 8, n.° 4, Outubro 1990, p. 387-430. Diponível em: <http://citeseer.nj.nec.com/straube90querie.html>

Tan, Kian-Lee; et Al. **Join and Muli-Join Processing in Data Integration Systems**. Data & Knowledge Engineering, Vol. 40, 2002, p. 217-239.

Templeton, M.; et. Al, **Mermaid: a front end to distributed heterogeneous databases**. IEEE Distributed Database System – Special Issue, n° 75, maio 1987, p. 695-708.

Wolski, A; Veijalainen, J. **Prepare and Commit Certification for Decentralized Transaction Management in Rigorous Heterogeneous Multidatabases**. Proceedings of the 8th International Conference on Data Engineering, Janeiro 1992, p. 470-479. Disponível em: <http://citeseer.nj.nec.com/veijalainen92prepare.html>