Uso de XML para Interoperabilidade entre Bases Heterogêneas

Juliana Pasqual, Marcos Sunye

Departamento de Informática, Universidade Federal do Paraná – UFPR, Curitiba –Paraná – Brasil.
Email: {jpasqual@inf.ufpr.br, sunye@inf.ufpr.br}

Resumo

A coexistência de banco de dados heterogêneos, implementados sobre diferentes modelos de dados, com linguagens, representações e plataformas diversas, tornou natural o uso de banco de dados integrados para compartilhamento de informações. 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, este trabalho apresenta uma solução baseada em XML. O resultado deste trabalho é a decomposição de uma expressão SQL, apresentada ao esquema integrado, em expressões SQL correspondentes a cada esquema local. A solução apresentada compreende: 1) o mapeamento do esquema integrado em relação aos seus esquemas componentes; 2) um algoritmo baseado nas regras de integração, utilizadas para gerar o esquema integrado, e nas informações de cada esquema local; 3) a utilização da estrutura de modelo objeto/relacional, abrangendo os conceitos de classe, herança, atributos multivalorados e compostos.

Palavras Chave: banco de dados heterogêneos, interoperabilidade entre banco de dados, XML em banco de dados

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 integrated schema, into SQL expressions correspondents to each local schema. The solution presented proposes: 1) mapping from integrated schema based on its components schemas; 2) algorithms based on integration's rules, used to create the integrated schema, and based on informations in the mapping; 3) utilization the relational and object/relational model and the concepts of class, inheritance, multivalued and composed attribute.

Key Words: heterogeneous databases, database interoperability, XML and databases

1 Introdução

Diferentes culturas organizacionais, fusões de empresas ou até mesmo falta de planejamento contribuem para o surgimento de diversos banco de dados em uma organização. Na maioria das vezes, estes bancos de dados de dados são heterogêneos e implementados utilizando diferentes modelos de dados, com linguagens, representações, tecnologias e plataformas diversas, que muitas vezes modelam objetos idênticos ou similares. A coexistência destes diferentes bancos de dados tornou-se natural e inevitável, porém surgiu a necessidade de compartilhamento destas informações, tanto para tomada de decisão, 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 banco de dados de forma transparente, como também manipular seus dados, podendo alterá-los, excluí-los e até mesmo incluir novos registros. Estes mecanismos devem também preservar a autonomia local dos banco de dados, mantendo sua integridade e coerência.

Com o objetivo de suprir a necessidade de interoperabilidade entre banco de dados heterogêneos, muitas linhas de pesquisas se destacaram. Uma delas propõem 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 banco de dados subjacentes, com cada um de seus objetos devidamente relacionados, via mapeamento, aos objetos correspondentes nos banco 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 banco de dados, nem da localização dos objetos neste.

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 banco de dados heterogêneos [1]. 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, não suportam todas as funcionalidades de um banco de dados múltiplo e não controlam transações quando trata-se de plataformas diferentes.

Para que a autenticidade e a coerência nos diversos banco 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 destes em todos os banco 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 banco de dados envolvidos, isto é, os banco 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 a 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.

O mapeamento será expresso através da linguagem de marcação eXtensible Markup Language, XML [2], 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.

As expressões serão escritas na linguagem SQL, Structured Query Language, utilizada pela grande maioria dos Sistemas de Gerenciamento de Banco de Dados, SGBDs, atuais. E também devido ao fato da nova especificação da SQL [3] possuir alguns recursos que abrangem as características do modelo objeto/relacional.

O trabalho segue, como padrão de entrada para as expressões, um sistema de visualização de esquemas – o VIQUEN, Visual Query Enviroment [4]. O VIQUEN é uma aplicação de consulta visual que opera sobre esquemas relacionais e que utiliza o modelo semântico 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.

2 Algoritmo e Mapeamento das Consultas

A decomposição de uma operação global em sub-operações é necessária devido a tecnologia utilizada para criar esquemas integrados estar baseada no mapeamento de visões e as técnicas para alterações em visões, ao longo da história, terem se deparado 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 exemplos, 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, 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 banco de dados envolvidos.

A proposta deste trabalho 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 banco de dados locais envolvidos na integração, para que sejam submetidas a estes bancos locais. A Figura 1 apresenta a arquitetura básica deste modelo.

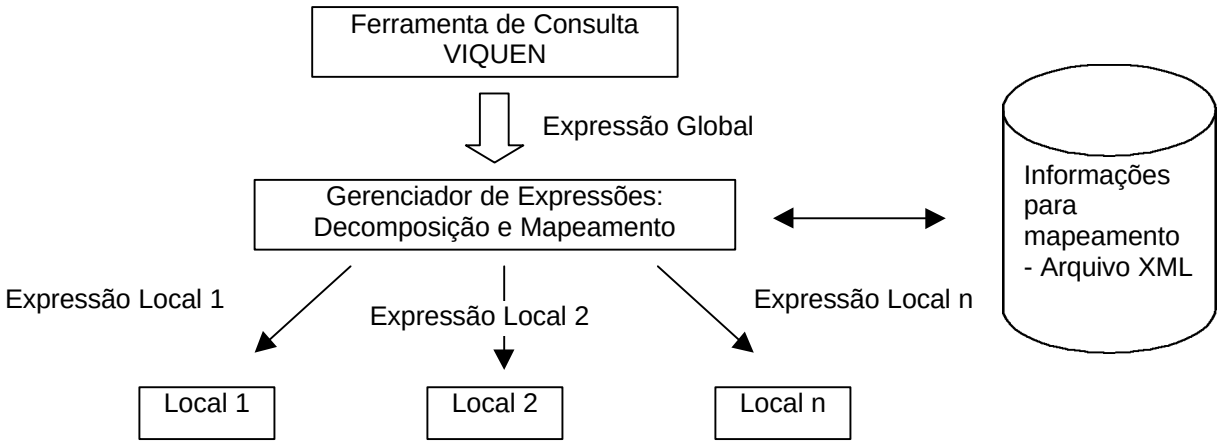

FIGURA 1 – ARQUITETURA DO PROTÓTIPO

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, deste que esteja no formato exigido pelo algoritmo. 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.

2.1 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 eles e ser possível o acesso integrado, suas diferenças devem ser resolvidas.

Este trabalho propõem 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.

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 –, cada uma de suas entidades, com informações dos seus banco de dados de origem, sua localização, a descrição e a função de mapeamento de seus atributos.

A Figura 2 apresenta a especificação formal do documento XML, utilizando para isso o padrão DTD. 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 mais antigo, 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 5 apresenta um fragmento do documento XML gerado sobre o esquema integrado apresentado no exemplo 2.1

```
<! -- 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 2 – ARQUIVO DTD: ESTRUTURA DO MAPEAMENTO PROPOSTO

Exemplo 2.1: 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 3 mostra os esquemas locais, a Figura 4 a estrutura do modelo integrado e a Figura 5 o mapeamento destes, segundo modelo DTD apresentado.

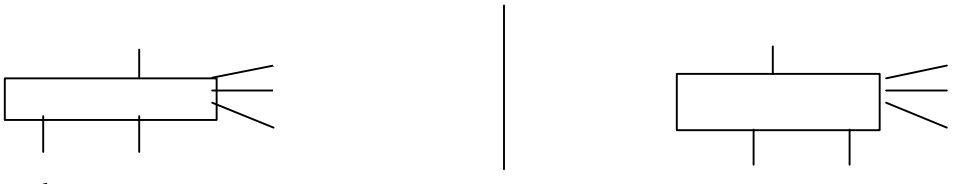

FIGURA 3 – DIAGRAMA DOS ESQUEMAS LOCAIS DO EXEMPLO 2.1

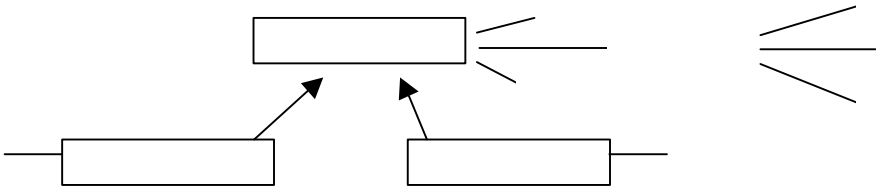

FIGURA 4 – DIAGRAMA DO ESQUEMA INTEGRADO DO EXEMPLO 2.1

```
<! -- 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>
    <atributo>
        <nome> telefone.celular <\nome>
        <atrib_componente objeto “Usuarios_bib” regra = “interseção”>
            <nome> celular <\nome>
            <mapeamento>  <função> f(x) = x <\função>  <\mapeamento>
        <\atrib_componente>
        <atrib_componente objeto “Empregados” regra = “interseção”>
            <nome> fone#1 <\nome>
            <mapeamento> <função> f(x) = x <\função> <\mapeamento>
        <\atrib_componente>
    <\atributo>
   <atributo>
        <nome> telefone.residencial <\nome>
        <atrib_componente objeto “Usuarios_bib” regra = “interseção”>
             <nome> residencial <\nome>
            <mapeamento>  <função> f(x) = x <\função>  <\mapeamento>
        <\atrib_componente>
        <atrib_componente objeto “Empregados” regra = “interseção”>
             <nome> fone#2 <\nome>
            <mapeamento>  <função> f(x) = x <\função>  <\mapeamento>
        <\atrib_componente>
    <\atributo>
    <atributo>
        <nome> telefone.comercial<\nome>
        <atrib_componente objeto “Usuarios_bib” regra = “interseção”>
             <nome> comercial <\nome>
            <mapeamento>  <função> f(x) = x <\função>  <\mapeamento>
        <\atrib_componente>
        <atrib_componente objeto “Empregados” regra = “interseção”>
             <nome> fone#3 <\nome>
            <mapeamento>  <função> f(x) = x <\função>  <\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 5 – MAPEAMENTO EM XML DO ESQUEMA INTEGRADO DO EXEMPLO 2.1

2.2 Algoritmo para Decomposição de Operações Globais

Devido a heterogeneidade e autonomia dos banco 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 consulta padrão e utilizada pela grande maioria dos SGBDs encontrados, hoje, no mercado.

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. Suas funções devem utilizar os padrões criados pela W3C, como o Document Objet Model, para acessar os dados em uma estrutura XML. Os algoritmos mostram as expressões de entrada sempre com o operador "and" a título de ilustração, porém isto não impede que elas possuam outros operadores da linguagem SQL, por exemplo o operador "or".

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
        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 E1 (e1, ..., en ) values ( 'x1', ..., 'xn')
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
```

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

2.3 Quanto as Regras de Integração e Mapeamento

Exemplo 2.2: Neste exemplo serão mostrados os principais passos dos três algoritmos apresentados. Considerando o mapeamento da Figura 5.

1º Caso – Algoritmo1

Expressão de entrada: Update pessoa Set escolaridade = 2 Where RG = '123.456-90'

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, *] = {" '123.456-90' "}

Where[1, *] = { "Doc_identificacao"}
ValorW[1, *] = {" '123.456-90' "}

Expressão = { "Update BD01.Usuario_bib set graduação = 2 where RG = '123.456-90' ", "Update BD02.Empregados set grau_escolaridade = 'terceiro grau' where Doc_identificacao = '123.456-90' }

2º Caso – Algoritmo1
Expressão de entrada: Update pessoa Set escolaridade = 5 Where RG = ‘123.456-90’
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, *] = {“ ‘123.456-90’ ”}

Where[1, *] = { “Doc_identificacao“}
ValorW[1, *] = {“ ‘123.456-90’ ”}

Expressão = { “Update BD01.Usuario_bib set graduação=2 where RG=‘123.456-90’“, “ERRO”}

3º Caso – Algoritmo2
Expressão de entrada: Delete from pessoa Where RG = ‘123.456-90’
Erro – Regra de equivalência não permite exclusão

4º Caso – Algoritmo3
Expressão de entrada: Insert into pessoa (RG, escolaridade) values (123.456-90, 3)
Erro – Regra de equivalência não permite inclusão

5º Caso – Algoritmo2
Expressão de entrada: Delete from Usuários_bib Where RG = ‘123.456-90’
Banco_dados = {“BD01”}
Obj_Original = {“Usuarios_bib”}

Where[0, *] = { “RG “}
ValorW[0, *] = {“ ‘123.456-90’ ”}

Expressao = { “Delete from BD01.Usuario_bib where RG = ‘123.456-90’ “}

6º Caso – Algoritmo3
Expressão de entrada: Insert into Empregados (RG, Data_admissão) values (‘123.456-90’, ‘01/02/2002’)
Banco_dados = { “BD02”}
Obj_Original = {“Empregados”}

Atributo[0,*] = {“Doc_identificação, Data_ admissão ”}
ValorA[0, *] = { ‘123.456-90’, ‘01/02/2002’}

Expressão =: { “ Insert into Empregados (Doc_ identificação, Data_admissão) values (‘123.456-90’, ‘01/02/2002’) “}

No 2º 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 3º e 4º Caso, como a regra de equivalência é Usuários_bib ñ 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 5º e 6º caso.

3 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 informações entre diferentes bases de dados e uma solução muito utilizada são os banco 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 armazenamento de informações, obter uma estrutura leve, padronizada e independente de plataformas, 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.

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.

Referências

[1] 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.º 4, Abril 1989.

[2] Normas Brasileiras de Padronização. **Linguagens de Banco de Dados – SQL**. Documento: NBR ISO/IEC 9075, 2000.

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

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

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

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

[7] 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.

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

[9] 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>

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

[11] **Extensible Markup Language**. Word Wide Web Consortium (W3C). Disponível em: <http://www.w3c.org/XML>.

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

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

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