Alexandre Tagliari Lazzaretti

XDC: Uma Proposta de Controle de Restrições de Integridade de Domínio em Documentos XML

Florianópolis – SC
2005

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

Alexandre Tagliari Lazzaretti

XDC: Uma Proposta de Controle de Restrições de Integridade de Domínio em Documentos XML

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

Prof Dr Ronaldo S. Mello
orientador

Florianópolis, Fevereiro de 2005.

XDC: Uma Proposta de Controle de Restrições de Integridade de Domínio em Documentos XML

Alexandre Tagliari Lazzaretti

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

_____

Raul Sidnei Wazlawick, Dr.

Banca Examinadora

_____

Ronaldo dos Santos Mello, Dr. (orientador)

_____

Frank Augusto Siqueira, Dr.

_____

Helena Grazziotin Ribeiro, Dra.

_____

José Leomar Todesco, Dr.

Agradecimentos

- Em primeiro lugar gostaria de agradecer a Deus, por ter saúde nesta caminhada, e a minha mãezinha querida Nossa Senhora Aparecida, que é a luz da minha vida, a qual ilumina e protege meus passos, em todos momentos da minha vida, muito obrigado.

- Gostaria de agradecer ao meu pai Alcides Lazzaretti, sempre no seu jeito calmo, nunca se envolvendo muito, mas sempre preocupado comigo, e a minha mãe Nahyr Tagliari Lazzaretti que ficava me esperando voltar das viagens semanais, com sua comidinha deliciosa e sua palavra de conforto, lhes admiro muito, tudo que sou na vida é graças ao que me ensinaram, amo vocês, muito obrigado;

- Gostaria de agradecer a minha esposa Tatiana, minha companheira, meu anjo da guarda, minha alma gêmea, meu porto seguro. Muito obrigado meu amor, saibas que tens muita participação neste trabalho, passei momentos muito difíceis, mas você sempre estava ao meu lado, me ajudando, me dando força, te admiro muito e sou muito feliz por ter uma grande Mulher como você ao meu lado, te amo, beijos;

- Gostaria de agradecer ao meu orientador Ronaldo dos Santos Mello, primeiramente pela confiança depositada, por sua disponibilidade, suas sugestões sempre válidas, e pela sua amizade, muito obrigado;

- Gostaria de agradecer minha família, que ficou sempre torcendo, aos meus irmãos Vana, Nêne e Nilton, meus sobrinhos queridos Levington (parceiro de algumas viagens) Antoniella, Fê, Glória, Laura, meu cunhado Lafayete e a minha cunhada Mabel, muito obrigado;

- Gostaria de agradecer a dois grandes amigos, Marcos Vinícius Lucatelli e Alexandre Lazzaretti Zanatta, exemplos de pessoas, com os quais aprendi muito, tanto na área do saber quanto no viver. Gostaria de colocar uma frase que foi o lema do nosso mestrado, meu e o do Zanatta, quando começamos a viajar 1.040 km toda a semana, “vamos ver o lado bom das coisas”, valeu Zapatta!

- Gostaria de agradecer a diretoria do Hospital Prontoclínicas, Julmar Biancini, Miriam Biancini e Bianca Biancini, por entenderem a minha necessidade e permitirem a conciliação da realização do mestrado com as atividades de trabalho. Muito obrigado!

- Gostaria de agradecer a todas as pessoas que me ajudaram de uma forma ou de outra para a realização desse trabalho. Foram tantos que não citarei nomes para não cometer injustiças, muito obrigado a todos!

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

API	Application Programming Interface
DOM	Document Object Model
DTD	Document Type Definition
EBNF	Extended Backus Naur Form
ECA	Event Condition Action
SAX	Simple API for XML
SQL	Structred Query Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
XDC	XML Domain Constraint
XDCL	XML Domain Constraint Language
XML	Extensible Markup Language
XSD	XML Schema Definition
XSL	XML Stylesheet Language
XSLT	XML Stylesheet Language Transform

Resumo

XML (*eXtensible Markup Language*) vem se consolidando como um padrão para exportação de dados entre aplicações na *Web*, por apresentar um formato textual simples e aberto. Essas características tornam-no adequado à representação de dados vindos de fontes heterogêneas. Restrições de integridade são mecanismos utilizados para a imposição de consistência em bancos de dados e também são utilizados em documentos XML.

Na especificação da tecnologia XML existem mecanismos para a imposição de restrições de integridade em documentos XML, as quais são definidas principalmente através do uso de esquemas XML: DTD (*Document Type Definition*) e XSD (*XML Schema Definition*). Nesta dissertação é definida uma nova classificação de restrições de integridade para o modelo relacional, baseada em classificações existentes na literatura, e um comparativo de como esquemas DTD e XSD as suportam. Conclui-se que esquemas XML (DTD e XSD) são insuficientes para garantir a totalidade de restrições de integridade existentes em bancos de dados relacionais.

Assim, esta dissertação define um controle de restrições de integridade de domínio, inexistente em esquemas DTD e XSD, para documentos XML e bancos de dados nativos XML. Esse controle é chamado XDC (XML *Domain Constraint*) e é composto por uma linguagem de incorporação de restrições de integridade chamada XDCL (*XML Domain Constraints Language*) e por um *parser* de validação de restrições chamado *parser* XDCL. Na linguagem XDCL é possível especificar condições e executar ações, semelhantemente à maneira como a linguagem SQL (*Structred Query Language*) trabalha com restrições de integridade no modelo relacional, através de gatilhos e *checks*. O *parser* XDCL é o mecanismo responsável por validar as condições e executar as ações, as quais foram definidas através do uso da linguagem XDCL e armazenadas num documento textual com extensão XDC.

Palavras-chave: Restrições de integridade, SQL, XML, XDC.

Abstract

XML (eXtensible Markup Language) has been raising as a standard to data interchange among applications in the Web, because it presents a simple text format. Because of these characteristics, it is suitable to represent data coming from heterogeneous sources. Integrity constraints are mechanisms used to guarantee consistency in databases and are also used in XML documents.

In the specification of XML technology there are mechanisms to specify integrity constraints in XML documents, which are defined mainly through the use of XML schemas: DTD (Document type definition) and XSD (XML Schema Definition). In this dissertation, one new classification of integrity constraints for the relational model is defined, based on existing classifications in the literature, and a comparative analysis of how schemas XML (DTD and XSD) support them. The conclusion is that XML schemas (DTD and XSD) are insufficient to guarantee all kinds of integrity constraints that may be defined in relational databases.

Thus being, this dissertation defines a control for domain integrity constraints that does not exists in schemas DTD and XSD. It may be applied to applications that manage XML documents, or native XML databases. This control is called XDC (XML Domain Constraint) and it is composed by a language that allows the definition of domain integrity constraints, called XDCL (XML Domain ConstraintsLanguage), and a parser that validates XDCL specifications, called XDCL parser. With XDCL it is possible to specify conditions and to execute actions, like SQL (Structured Query Language) does with integrity constraints in the relational model, by defining triggers and check clauses. The XDCL parser is responsable to validate the conditions and to execute the actions defined in a textual document with extension XDC. This document contains a set of XDCL instructions.

Keywords: Integrity constraints, SQL, XML, XDC.

1 INTRODUÇÃO

XML (*eXtensible Markup Language*) é um padrão atual para representação e intercâmbio de dados na *Web* (W3CXML, 2004). Um conjunto de dados XML é descrito num documento XML. Do ponto de vista de banco de dados, um documento XML é uma coleção de dados, da mesma forma que um banco de dados. Porém, a tecnologia XML não é equivalente à tecnologia de banco de dados, pois não existem soluções para todos os aspectos de gerenciamento de dados, como controle de restrições de integridade, gerenciamento de transações, indexação e consultas a múltiplos documentos (CHAUDHRI *et al*, 2003).

Um aspecto importante, citado anteriormente, é o controle de restrições de integridade. Um dos objetivos primordiais de um sistema gerenciador de banco de dados é a integridade dos dados. Dizer que os dados num banco de dados estão íntegros significa dizer que refletem a realidade representada pelo banco de dados e que são consistentes entre si (HEUSER, 2001; DATE, 2000; ELMASRI & NAVATHE, 2000; SILBERCHATZ *et al.*, 1999). Para garantir a integridade de um banco de dados são necessárias regras que estabeleçam as consistências dos dados no banco de dados, chamadas "regras de integridade". Existem várias categorias de restrições de integridade. No modelo relacional, podem ser classificadas como restrições de *domínio*, de *chaves*, de *integridade referencial*, *quanto ao momento de verificação* e *baseada em eventos*, sendo que as categorias de *domínio* e *chaves* possuem subdivisões. A categoria de *domínio* define quais os possíveis valores que um atributo deve ter e subdivide-se em *atributos*, *tipo*, *tuplas*, *banco de dados* e *transição de estados*. A categoria *chaves* subdivide-se em *candidatas*, *primárias* e *estrangeiras*.

A linguagem SQL é a linguagem padrão para o tratamento das restrições de integridade do modelo relacional, citadas anteriormente, pois possui recursos para o

controle de todas as categorias. Com relação a restrições de domínio, a SQL possui cláusulas como gatilhos e *checks* que permitem a criação de restrições elaboradas e completas.

Assim como nos sistemas de bancos de dados, documentos XML também necessitam de regras que estabeleçam a integridade dos dados. Na tecnologia XML existem as especificações de esquemas XML (DTD e XSD), os quais permitem definir a estrutura, o conteúdo e a semântica de documentos XML. Porém, elas são insuficientes para atender a todas categorias de restrições de integridade existentes nos modelos de bancos de dados, mais especificamente, no modelo relacional.

1.1 Hipóteses da pesquisa

Esta dissertação busca tratar a categoria de restrições de integridade de domínio para documentos XML, visando suprir as lacunas existentes na especificação dos esquemas XML, para o tratamento dessa categoria de restrições de integridade.

Pretende-se criar um controle, o qual é composto por uma linguagem para especificação de restrições de integridade e um mecanismo para a sua imposição em documentos XML. A linguagem de especificação de restrições de integridade usa como base a maneira que a linguagem SQL trata as restrições de integridade de domínio em bancos de dados relacionais.

1.2 Objetivos

Objetivo geral

Propor uma abordagem para controle de restrições de integridade de domínio em documentos XML, chamada controle XDC (*XML Domain Constraint*). Esta abordagem visa suprir as deficiências para imposição de restrições de integridade de domínio encontradas nos esquemas XML e em trabalhos relacionados.

Objetivos específicos

Os objetivos específicos deste trabalho são:

- estabelecer uma nova classificação de restrições de integridade para o modelo relacional: por meio da análise das obras de diversos autores, verificou-se que as classificações de restrições de integridade para o modelo relacional não são homogêneas; portanto, existe a necessidade de ter uma classificação única, que incorpore todas as obras estudadas e atenda a todas as necessidades de imposição de restrições;

- criar uma linguagem e um mecanismo de validação desta: o controle XDC é composto por uma linguagem de definição de restrições chamada linguagem XDCL (XDC *Language*) e por um *parser* de validação de restrições chamado *parser* XDCL.

- ter gerência independente de integridade: o controle XDC é um tratamento adicional e independente da arquitetura de validação de dados XML, complementando os controles de restrições de integridade já disponíveis nos esquemas XML. Seu gerenciamento é independente, pois não há a necessidade de adicionar elementos ou atributos em esquemas XML para que a uma validação de integridade seja feita. Controles XDC são definidos em documentos XML específicos e tratados por um *parser* específico;

- possuir sintaxe XML: a linguagem XDCL é composta por um conjunto de elementos e atributos organizados hierarquicamente numa estrutura XML. Assim, especificações de integridade são definidas através de documentos XML e os recursos da tecnologia XML podem ser usados para validar e processar esses documentos;

- tratar de restrições de integridade inexistentes na especificação de esquemas XML: o controle XDC impõe as restrições de integridade classificadas como de tuplas, banco de dados e transição de estados, cujo tratamento não existe em esquemas XML. Portanto, o uso conjunto do controle XDC e de um esquema XML permite que todas as subcategorias de restrições de integridade de domínio sejam impostas.

- usar como base a maneira que a linguagem SQL trata as restrições de integridade em bancos de dados relacionais, utilizando características semelhantes a gatilhos e *checks*.

1.3 Justificativas

O foco desta dissertação é a imposição de restrições de integridade em documentos XML, mais precisamente, as classificadas no modelo relacional como restrições de integridade de domínio, visto que esquemas XML não possuem recursos suficientes para tratamento de todas as subcategorias de restrições de integridade de domínio, abrangendo somente a subcategoria classificada como tipo e, parcialmente, o tratamento de restrições de integridade da subcategoria atributos, não considerando a categoria de restrições de integridade de transição de estados. Alguns trabalhos relacionados a este tópico são encontrados na literatura (OGBUJI 2001, PROVOST 2002, BENEDIKT *et al.* 2002, BAYLEY *et al.* 2002). De modo geral, todos propõem uma maneira de especificar restrições de integridade e um mecanismo para impô-las. Mesmo trazendo contribuições para este tópico de pesquisa, esses trabalhos apresentam algumas deficiências, a saber:

- nenhuma das propostas considera o tratamento de restrições de integridade de transição de estados, parte importante integrante das restrições de integridade de domínio;

- nenhum dos trabalhos utiliza os recursos da linguagem SQL como referência, sendo SQL uma linguagem consolidada para o tratamento de restrições de integridade em bancos de dados relacionais;

- nenhum dos trabalhos relacionados propõe uma abordagem expressiva o suficiente para impor restrições de integridade elaboradas para documentos XML.

Portanto, esta dissertação propõe um controle de restrições de integridade de domínio em documentos XML, inexistente na especificação de esquemas XML (DTD – *Document Type Definition* e XSD – *XML Schema Definition*), baseando-se em recursos semelhantes aos existentes na linguagem SQL. A escolha da categoria de restrições de

integridade de domínio justifica-se pela necessidade de controles mais completos para essa categoria em documentos XML e, também, para evitar que a proposta da dissertação se tornasse complexa, caso fossem tratadas todas as categorias de restrições de integridade do modelo relacional.

1.4 Organização dos capítulos

O presente trabalho apresenta outros quatro capítulos. O capítulo 2 é dividido em três seções. A primeira apresenta uma classificação de restrições de integridade no modelo relacional; a segunda, um estudo do modo como a linguagem SQL impõe restrições de integridade; a terceira mostra um estudo da tecnologia XML no qual são apresentados alguns padrões utilizados no desenvolvimento deste trabalho.

O capítulo 3 é dividido em duas seções. Na primeira, são mostrados três comparativos: o primeiro entre as restrições de integridade do modelo relacional com os mecanismos de imposição dessas pela linguagem SQL; o segundo, entre as restrições de integridade do modelo relacional com os mecanismos de imposição de restrições de integridade da DTD e o terceiro, entre restrições de integridade do modelo relacional com os mecanismos de imposição de restrições de integridade do XSD. Com base nesses comparativos são levantadas as necessidades dos esquemas DTD e XSD na imposição de restrições de integridade e verificado como a linguagem SQL impõe as restrições de integridade do modelo relacional. Na segunda seção é feito um levantamento de trabalhos relacionados que buscam suprir as necessidades citadas na seção anterior, principalmente com relação a restrições de integridade de domínio. Dois comparativos são feitos: os trabalhos relacionados com relação às categorias de restrições de integridade de domínio do modelo relacional e os trabalhos relacionados com relação à linguagem SQL. Esses comparativos estabelecem a motivação para a proposta da dissertação.

O capítulo 4 é dividido em quatro seções: na primeira, é apresentada a arquitetura da abordagem XDC, mostrando como esta se insere no contexto de processamento de dados XML; na segunda, é feita uma descrição da linguagem XDCL; na terceira, é apresentado o *parser* XDCL e, na quarta e última, é mostrada a aplicação da linguagem XDCL e do parser XDCL num estudo de caso.

Finalmente, no capítulo 5 são apresentadas as contribuições da dissertação, as considerações finais e as atividades futuras relacionadas.

2 MÉTODOS E ÁREAS RELACIONADAS

Este capítulo apresenta os métodos utilizados para o desenvolvimento da dissertação e o estudo feito sobre as áreas relacionadas, em específico, restrições de integridade em bancos de dados relacionais, linguagem SQL e tecnologia XML, mostrando seus conceitos, classificações e de que maneira podem ser definidas.

2.1 Métodos

Para desenvolver o controle de restrições de integridade de domínio para documentos XML desta dissertação, realizou-se, primeiramente, um estudo do modelo relacional com o objetivo de definir uma classificação para as categorias de restrições de integridade. Este estudo se justifica pelo fato de este modelo ser consolidado na comunidade de banco de dados e possuir uma classificação de restrições de integridade abrangente.

Em seguida realizou-se um estudo da linguagem SQL, com o objetivo de levantar quais controles são utilizados para o tratamento de restrições de integridade em bancos de dados relacionais. Sua importância se justifica pelo fato de a linguagem SQL ser a linguagem padrão para o tratamento de restrições de integridade em bancos de dados relacionais. Após, é realizado um estudo da tecnologia XML, com o objetivo de levantar quais controles existem e quais não existem, para a imposição de restrições de integridade em documentos XML.

Feito o levantamento das carências existentes na tecnologia XML para imposição de restrições de integridade, realizou-se um estudo de trabalhos relacionados, existentes na literatura, que visam suprir essas carências. Nesses trabalhos são encontradas algumas deficiências, as quais dentre outros motivos, serviram como base para a

realização desta dissertação. Os estudos citados são mostrados detalhadamente a seguir na seção 2.2.

2.2 Áreas relacionadas

Esta seção mostra os estudos feitos no modelo relacional, linguagem SQL, tecnologia XML e trabalhos relacionados, os quais fazem parte do escopo da dissertação.

2.2.1 Restrições de integridade em bancos de dados relacionais

O termo "integridade" refere-se à *precisão* ou *correção* de dados no banco de dados. Assim, as regras de integridade fornecem a garantia de que mudanças feitas no banco de dados por usuários autorizados não resultem em perda da consistência de dados, protegendo o banco de dados de danos acidentais (DATE, 2000, SILBERCHATZ *et al.*, 1999).

Caso as restrições de integridade dos dados no banco de dados não sejam garantidas, têm-se valores não desejados, valores desconhecidos ou nulos e relacionamentos perdidos ou incorretos. Quando uma nova restrição de integridade é declarada, o banco de dados deve efetuar uma validação e aceitar ou não a restrição de integridade. Uma vez aceita, a restrição de integridade é verificada pelo sistema gerenciador do banco de dados toda vez que uma modificação é feita nos dados aos quais ela se refere.

Existe uma grande variedade de restrições de integridade em bancos de dados relacionais. DATE (2000) classifica-as como: restrições de tipo (domínio), restrições de atributo, restrições de variáveis de relação (tuplas), restrição de banco de dados, restrições de transição de estados, restrições de chaves, restrições de integridade referencial e restrições quanto ao momento de verificação. Já SILBERCHATZ *et al.* (1999) classificam-nas como restrições de domínio, restrições de chaves, restrições de formas de relacionamentos e restrições de integridade referencial. ELMASRI & NAVATHE (2000) classificam as restrições de integridade da seguinte maneira: restrições de domínio, restrições de chave, restrições de entidade e restrições de integridade referencial. HEUSER (2001) classifica as restrições de integridade como:

integridade de domínio, integridade de vazio (se o atributo é obrigatório ou opcional), integridade de chaves e integridade referencial.

Com base nas classificações dos autores citados acima, pode-se concluir que as classificações existentes na literatura são, em alguns casos, conflitantes. Portanto, esta dissertação define uma nova classificação de restrições de integridade, conforme segue: restrições de domínio (DATE, 2000; SILBERCHATZ *et al.*,1999; ELMASRI & NAVATHE, 2000; HEUSER, 2001), restrições de chaves (DATE, 2000; SILBERCHATZ *et al.*,1999; ELMASRI & NAVATHE, 2000; HEUSER, 2001), restrições de integridade referencial (DATE, 2000; SILBERCHATZ *et al.*,1999; ELMASRI & NAVATHE, 2000; HEUSER, 2001) e restrições quanto ao momento de verificação (DATE, 2000). Além dessas, definiu-se uma nova categoria, denominada de restrições baseadas em eventos, que são restrições que podem ser impostas através da chamada de um procedimento por uma aplicação, ou de através da execução de um gatilho por um evento ocorrido no banco de dados. Essa classificação é vista com mais detalhes a seguir.

a) Restrições de domínio

Restrições de domínio são as mais elementares formas de restrições de integridade, sendo verificadas toda vez que um item é incorporado ou modificado no banco de dados. Essas restrições não somente verificam os valores inseridos ou modificados no banco de dados, mas também testam consultas para garantir que comparações feitas tenham sentido. ELMASRI & NAVATHE (2000) definem restrições de domínio como a especificação de quais valores cada atributo deve ter. Isso pode ser feito através de especificação de tipos de dados associados aos domínios, ou através da definição de valores válidos para o tipo, ou ainda, através de instruções que verificam quais são os valores válidos para este tipo. Portanto, as restrições de domínio podem ser classificadas da seguinte maneira:

- restrições de atributo: é (ou é logicamente equivalente a) apenas uma enumeração de seus valores válidos. Como exemplo, define-se um atributo PESO e impõe-se uma restrição que o atributo PESO deve ser maior que zero. Então, qualquer expressão

de atualização que for avaliada para esse atributo deverá obedecer a essa restrição; caso contrário, irá falhar.

- restrições de tipo: é apenas uma declaração para o efeito de que um atributo seja de um tipo especificado, ou seja, as restrições de tipo fazem parte da definição do atributo em questão e podem ser identificadas por meio do tipo correspondente. Por exemplo, na definição do atributo VDATA (DATE), o atributo VDATA possui o seu valor limitado ao tipo DATE.

- restrições de tupla: é uma restrição sobre uma tupla individual, podendo incorporar restrições sobre vários atributos da mesma. As restrições de tupla geralmente são executadas logo após qualquer instrução que possa fazer com que elas sejam violadas. Por exemplo, uma pessoa só poderá passar para o estado civil 'casado' se a sua idade for superior a 18 anos.

- restrições de banco de dados: é uma restrição que envolve duas ou mais tuplas distintas. Neste caso, é necessário que todas as restrições de tupla envolvidas sejam atendidas; caso contrário, a operação não será efetivada. Por exemplo, o somatório dos valores de um atributo quantidade de uma tabela itens de pedido não pode ser maior que o valor do atributo quantidade total de uma tabela de pedidos.

- restrições de transição de estado: estão relacionadas com os estados corretos dos valores dos atributos do banco de dados, garantindo que a transição de um estado correto para outro seja válida. Por exemplo, num banco de dados de pessoas, pode haver uma série de restrições de transição relacionadas à mudança de estado civil. A passagem do estado SOLTEIRO para CASADO é válida, porém não é válida a passagem do estado SOLTEIRO para VIÚVO.

b) Restrições de chaves

Segundo DATE (2000), o modelo relacional sempre enfatizou o conceito de chaves, embora seja na realidade apenas um caso especial de restrições. Chave é o conceito básico para identificar linhas e estabelecer as suas relações em bancos de dados

relacionais (HEUSER, 2001). Podem-se classificar as chaves como candidatas, primárias ou alternativas e estrangeiras:

- chaves candidatas: seja X um conjunto de atributos de uma tupla Y. X é uma chave candidata se obedecer às propriedades de unicidade e irredutibilidade. A propriedade de unicidade é estabelecida quando nenhum valor válido de Y contém duas tuplas com o mesmo valor de X. A irredutibilidade é dada quando nenhum subconjunto de X tem a propriedade da unicidade, ou seja, as chaves candidatas não devem incluir atributos que sejam irrelevantes para fins de identificação única. Pode ser que exista mais de uma chave candidata em uma tupla;

- chaves primárias ou alternativas: no caso de existir mais de uma chave candidata, o modelo relacional exige que, pelo menos, uma dessas chaves seja definida como chave primária. As outras são chamadas, automaticamente, de chaves "alternativas";

- chaves estrangeiras: é um conjunto de atributos de uma tupla cujos valores devem, obrigatoriamente, corresponder a valores de alguma chave primária de uma outra ou da mesma tupla.

Com base nas restrições de chaves, alguns pontos importantes podem ser observados: (i) uma chave estrangeira deve possuir um valor correspondente ao de uma chave primária. Já o contrário não é verdadeiro; (ii) uma chave estrangeira é simples ou composta se a chave primária é simples ou composta; (iii) cada atributo da chave estrangeira deve possuir o mesmo tipo de dado do atributo correspondente da chave primária associada.

c) Restrições de integridade referencial

O problema de garantir que o banco de dados não inclua valores inválidos de chave estrangeira é conhecido como o problema de integridade referencial.

A integridade referencial garante que o banco de dados não deve conter quaisquer valores de chaves estrangeiras não associados a uma chave primária, ou seja, se B faz relação a A, então A deve existir.

d) Restrições quanto ao momento da verificação

Outra classificação de restrições de integridade em bancos de dados é quanto à questão de quando a verificação é feita. Geralmente, definem-se duas categorias de verificação: imediata e postergada.

No caso imediato, a verificação é feita no momento da ocorrência de uma operação, ou seja, imediatamente. Já no caso postergado, as restrições são verificadas no momento que a instrução COMMIT[2] é executada ou em algum momento posterior a execução da operação, sendo esse momento definido pelo usuário. Caso a restrição seja violada, a operação não é executada. Por exemplo, quando se necessita alterar os valores de um determinado atributo código que é chave primária, soma-se a eles um valor em todas as tuplas da relação. Essa operação só poderá ser efetivada no momento que todos os valores do atributo código forem alterados, para não haver problema de violação de unicidade na relação durante o processo. Se não houver violações, executa-se o COMMIT; do contrário, a operação é desfeita.

e) Restrições baseadas em eventos

Existem restrições de integridade que podem ser programadas pelo usuário, e cuja verificação é independente da execução de operações de atualização no banco de dados. Essas restrições são denominadas restrições de integridade baseadas em eventos. Neste caso, as restrições são verificadas através de uma chamada da aplicação. Um exemplo é um procedimento que efetua o pagamento automático de uma conta, ou seja, se o dia do vencimento da conta é igual a um determinado dia do mês, então a conta é paga e as atualizações de saldo são feitas automaticamente no banco de dados.

No caso de a verificação ser dependente de atualizações no banco de dados, as restrições são impostas pelo SGBD (Sistema de Gerência do Banco de Dados) e o recurso que geralmente é utilizado para implementar esse tipo de operação são os gatilhos. Gatilhos são comentados na próxima seção.

[2] COMMIT indica o término de uma operação bem-sucedida. Todas as operações feitas são gravadas.

2.2.2 Restrições de integridade em SQL

SQL é a linguagem padrão para banco de dados relacionais. Segundo DATE (2000), as restrições de integridade em SQL podem ser classificadas nas seguintes categorias: restrições de domínios, restrições básicas de tabela, restrições gerais, gatilhos, procedimentos, transações e quanto ao momento de verificação.

- Restrições de domínios: É uma restrição que se aplica a cada coluna definida no domínio em questão. Sua definição compreende uma enumeração de valores que formam o domínio em questão. A restrição de domínio em SQL é definida através da cláusula *domain*. Um exemplo é mostrado na Figura 2.1, que define os valores válidos para o domínio estado_civil.

```
CREATE DOMAIN estado_civil CHAR(10)
        CONSTRAINT estados_validos
        CHECK (VALUE IN
        (‘Solteiro’, ‘ Casado’, ‘Viúvo’, ‘Desquitado’,’Divorciado’));
```

Fonte: Primária.

Figura 2.1- Exemplo de restrição de domínio.

- Restrições gerais: São definidas, por meio de um recurso chamado assertiva (*assertion*). Uma assertiva é um predicado que deve ser sempre verdadeiro; geralmente envolve dados de várias tabelas do banco de dados e independe da operação de atualização executada. A Figura 2.2 mostra um exemplo de declaração de uma assertiva na qual toda pessoa tem idade maior que zero.

```
        CREATE ASSERTION testaidade
        CHECK (
        (SELECT MIN (pessoas.idade) FROM PESSOAS)>0);
```

Fonte: Primária.

Figura 2.2- Exemplo de restrição geral.

- Restrições básicas de tabelas: Uma restrição básica de tabela pode ser: uma definição de chave candidata (definida através das cláusulas PRIMARY KEY e UNIQUE), uma definição de chave estrangeira (definida através da cláusula FOREIGN KEY) e uma definição de restrição de verificação (definida através da cláusula CHECK). A cláusula CHECK é utilizada para assegurar que os valores dos atributos possam satisfazer a determinadas condições. A Figura 2.3 mostra um exemplo envolvendo os três tipos de restrições básicas de tabelas.

```
CREATE TABLE pessoas
        (codigo integer not null,
        nome char(50) not null,
        cidade integer not null,
        ativo char (1) not null,
        idade integer not null,
        PRIMARY KEY (codigo),
        FOREIGN KEY (cidade),
        CHECK (idade>=0 and idade<=120));
```

Fonte: Primária.

Figura 2.3- Exemplo de restrições básicas de tabelas.

- Procedimentos: Procedimentos são programas escritos em SQL que são armazenados num banco de dados e são executados através de uma chamada de aplicação. São utilizados quando várias declarações SQL necessitam ser executadas simultaneamente, ou uma ou mais declarações ser executadas várias vezes. Portanto, a motivação do uso de procedimentos dá-se por modularização de funcionalidades, reaproveitamento de código e concentração de lógica de manipulação de dados no banco de dados.

Procedimentos podem ser usados para especificar restrições de integridade baseadas em eventos, onde o evento é a chamada do procedimento pela aplicação. Por exemplo, a Figura 2.4 mostra um procedimento que é executado toda vez que é processado um arquivo de pagamentos efetuados pelos clientes. São passados como parâmetros o número da fatura (numero) e o valor pago (val), sendo o valor pago (contas.valpago) das faturas atualizado no banco de dados.

```
CREATE PROCEDURE BaixaContas (numero long, val float)
AS
    BEGIN
        UPDATE contas
        SET contas.valpago=:val
        WHERE contas.numero=:numero
    END
END;
```

Fonte: Primária.

Figura 2.4- Exemplo de procedimento.

- Gatilhos: Um gatilho é um conjunto de instruções que é executado automaticamente para manter a integridade do banco de dados, em conseqüência da ocorrência de alguma operação de atualização no banco de dados (SILBERCHATZ *et al.*, 1999, DATE, 2000). Quando se define um gatilho, devem-se especificar as condições sobre as quais ele deve ser executado e as ações que serão tomadas quando ele for

disparado. A Figura 2.5, por exemplo, define um gatilho que tem a função de atualizar o atributo MAIORIDADE na tabela de PESSOAS com o valor “TRUE”, quando o atributo IDADE é maior ou igual a 18. Esta ação ocorre após a inserção de um novo código.

```
CREATE TRIGGER PESSOAS_AI ON PESSOAS
AFTER INSERT, UPDATE
BEGIN
        Update PESSOAS
        Set MAIORIDADE='TRUE'
        Where IDADE>=18;
END
```

Fonte: Primária.

Figura 2.5- Exemplo de gatilho.

- Transações: Um dos recursos fornecidos por bancos de dados é o controle de transações, que permite às aplicações manterem a integridade lógica dos dados em determinado processo.

Uma transação é um conjunto de comandos executados que incluem uma ou mais operações de acesso ao banco de dados que devem ser aplicadas integralmente. Se um dos comandos falhar, todos os outros precisam ser desfeitos para manter a integridade. Essas operações podem ser uma inclusão, uma alteração, uma exclusão ou uma recuperação (OLIVEIRA, 2000; ELMASRI & NAVATHE, 2000).

As operações que compõem as transações podem estar embutidas num programa de aplicação ou podem ser especificadas diretamente no banco de dados pelo usuário.

As transações devem transformar um estado consistente do banco de dados em outro estado consistente. Por exemplo, na tabela PESSOAS criada na Figura 2.2, suponha a criação de um atributo adicional NUMERO_SOCIOS. Uma pessoa pode ter vários sócios, tendo-se, então, outra tabela, chamada SOCIOS_PESSOAS. Toda vez que uma tupla for adicionada à tabela SOCIOS_PESSOAS, tem-se de atualizar o atributo NUMERO_SOCIOS da tabela PESSOAS. Ainda, toda vez que for excluída uma tupla da tabela SOCIOS_PESSOAS, deve-se atualizar o atributo NUMERO_SOCIOS da tabela PESSOAS.

As transações iniciam com a execução de uma instrução BEGIN TRANSACTION e terminam com a execução bem-sucedida de uma instrução COMMIT ou ROLLBACK. Uma instrução COMMIT encerra uma transação de forma bem-sucedida. Neste caso, o banco de dados está, ou deveria estar, consistente e todas as atualizações feitas pelas operações executadas nas transações podem se tornar permanentes. Em contraste, uma instrução ROLLBACK indica o término de uma transação malsucedida, ou seja, o banco de dados está em um estado inconsistente e todas as operações executadas nas transações devem ser desfeitas.

A Figura 2.6 mostra um exemplo de transação no qual, toda vez que se inclui uma tupla na tabela SOCIOS_PESSOAS, atualiza-se o atributo NUMERO_PESSOAS da tabela PESSOAS.

```
BEGIN TRANSACTION
   INSERT INTO socios_pessoas ( ...);
   UPDATE pessoas SET numero_socios=numero_socios+1
   WHERE pessoas.codigo=new.socio;
COMMIT TRANSACTION
END;
```

Fonte: Primária.

Figura 2.6- Exemplo de transação.

- Quanto ao momento de verificação: no que diz respeito ao momento que uma restrição de integridade é verificada, pode ser definida como DEFERRABLE ou NOT DEFERRABLE em SQL.

A restrição NOT DEFERRABLE é sempre verificada imediatamente. A restrição DEFERABLE pode ser definida como INITIALLY DEFERRED (estado em que a restrição não é verificada) e INITIALLY IMMEDIATE (estado em que a restrição é verificada). A verificação de uma restrição de integridade pode ser ativada e desativada dentro do corpo de uma transação por meio da instrução SET CONSTRAINTS <NOME_RESTRIÇÃO> <OPÇÃO>, onde <OPÇÃO> tem os valores DEFERRED ou IMMEDIATE. Por exemplo, pode-se definir que a restrição geral testaidade, definida na Figura 2.3, não seja verificada imediatamente através do seguinte comando: SET CONSTRAINTS testaidade DEFERRED.

2.2.3 XML

De acordo com o W3C[3], XML (*eXtensible Markup Language*) foi originalmente projetado para vir ao encontro dos desafios da publicação eletrônica em larga escala e, também, para ter um importante papel na exportação e troca de dados na *Web*.

XML é uma metalinguagem de marcação, ou seja, é composto de texto e marcação. É um padrão aberto onde cada aplicação define seu protocolo para representação dos dados. Ela utiliza delimitadores chamados *tags* para descrever seus dados. Uma *tag* indica a intenção do dado e delimita o seu conteúdo. Um dado em XML é chamado de "elemento" e seu conteúdo é delimitado por uma *tag* inicial (*Start Tag*) e uma *tag* final (*End Tag*).

Um conjunto de dados XML é descrito num documento XML. A Figura 2.7 mostra um exemplo de documento XML para o domínio de cadastro de pessoas. Conforme o exemplo, a organização dos dados em um documento XML é hierárquica, existindo sempre um elemento raiz, por exemplo, o elemento pessoas. Um elemento pode ser composto de um ou mais elementos, que são chamados de "elementos filhos", por exemplo, o elemento codigo. Os elementos filhos também podem conter uma hierarquia de elementos, por exemplo, os elementos nome, endereço, cidade, que, neste caso, são filhos do elemento codigo, o qual, por sua vez, é filho do elemento pessoa. Os elementos podem conter atributos, que são propriedades que se deseja manter sobre o elemento em questão. Por exemplo, na Figura 2.7 tem-se o atributo id que especifica uma identificação para o elemento codigo.

Um documento XML, no que se refere a restrições, pode ser classificado em *bem formado* e *válido*. Todos os documentos XML que estão de acordo com as especificações de sintaxe XML são ditos *bem formados.* Elementos, em documentos *bem formados*, são delimitados por uma *tag* inicial (*Start tag*) e uma *tag* final (*End tag*) e devem conter um único elemento pai, com exceção do elemento raiz (*root*). Também não existem referências a entidades externas, a menos que um esquema esteja definido para o documento XML.

[3] W3C (World Wide Web Consortium) é um consórcio formado por acadêmicos e empresários que buscam definições de padrões para *WEB*.

```
<? xml version = ”1.0” encoding  = ”UTF-8” standalone = “yes” >
<pessoas>
      <pessoa>
          <codigo id=”C1”>01</codigo>
          <nome>
              <primeiro> Maria </primeiro>
              <meio> Aparecida </meio>
              <sobrenome> Tavares </sobrenome>
          </nome>
          <endereco>
              <rua> Rua das Flores </rua>
              <numero> 1789 </numero>
              <bairro> Centro </bairro>
          </endereco>
          <cidade> Rio de Janeiro </cidade>
      </pessoa>
</pessoas>
```

Fonte: Primária.

Figura 2.7 - Exemplo de um documento XML

A outra forma de restrição de documentos XML refere-se a um documento *válido*. Um documento XML é dito *válido* se possui um esquema hierárquico definido (DTD ou XSD) e se a sua estrutura está de acordo com este esquema.

A validação de um documento XML em termos de *bem formado* e/ou *válido* é feita por programas chamados *parsers*.

Uma vez definido um documento estruturado em XML, é possível utilizar tecnologias específicas para manipulá-lo da forma que se deseja. Essas tecnologias são padrões criados pelo W3C para serem utilizados em documentos XML; cada uma delas possui sua especificação.

Dentre essas tecnologias têm-se os esquemas XML, os quais expressam vocabulários compartilhados, que permitem às máquinas carregar regras feitas por pessoas. Fornecem uma maneira de definição da estrutura, conteúdo e semântica de documentos XML (W3CXMLSchema, 2004).

O W3C recomenda dois tipos de esquemas, visando organizar ou definir elementos e atributos em documentos XML. São eles a DTD e o XSD, apresentados a seguir.

DTD

Na recomendação do W3C do XML 1.0, foi inserido a DTD (*Document Type Definition*) como a primeira tentativa para estabelecer regras para estruturar os documentos XML. PITTS-MOULTIS & KIRK (2000) definem: “Em termos genéricos, DTD é um arquivo que é separado do restante do documento XML principal e que fornece um conjunto de regras para o documento XML ao qual ela é anexada.”

As declarações são feitas através de palavras-chaves. A Tabela 2.1 mostra os tipos de declarações, com suas palavras-chaves e significados, que podem ser possíveis de ocorrer em uma DTD.

Tabela 2.1- Tipos de declarações de marcação

Palavra-Chave	**Significado**
ATTLIST	Declaração de atributos, que podem especificar um tipo de elemento e os valores permitidos para o atributo.
ELEMENT	Declaração de elementos.
ENTITY	Declaração de reuso de um conteúdo.
NOTATION	Formato de declaração para conteúdo externo.

Fonte: Primária.

No padrão DTD, os elementos são declarados através da palavra-chave <!ELEMENT, seguida pelo seu nome e tipo. Por exemplo, na Figura 2.8, as declarações dos elementos: pessoa, codigo, nome, endereco, cidade, primeiro, meio, sobrenome, rua, número e bairro.

```
<?xml version =“1.0” encoding =“UTF-8” standalone =”yes”>
    <!DOCTYPE  pessoas [
    <! ELEMENT pessoa (codigo+)>
    <! ELEMENT codigo (nome, endereco, cidade)*>
    <! ATTLIST codigo id CDATA>
    <! ELEMENT nome (primeiro, meio?, sobrenome)>
    <!ELEMENT primeiro (#PCDATA)>
    <!ELEMENT meio (#PCDATA)>
    <!ELEMENT sobrenome (#PCDATA)>
    <! ELEMENT endereco (rua, numero, bairro)>
    <!ELEMENT rua (#PCDATA)>
    <!ELEMENT numero (#PCDATA)>
    <!ELEMENT bairro (#PCDATA)>
    <!ELEMENT cidade (#PCDATA)>
    ]>
```

Fonte: Primária.

Figura 2.8- Exemplo de DTD.

Os elementos declarados podem ser de dois tipos: simples e complexos. Os elementos simples podem conter apenas dados de caracteres ou texto e nenhum outro elemento adicional, ou seja, não se pode aninhar outro elemento. Neste caso, o elemento

deve ser definido com tipo #PCDATA (*parsed caracter data*). Por exemplo, na Figura 2.8, os elementos primeiro, meio, sobrenome, rua, bairro e número são elementos simples.

Elementos complexos são elementos que contêm outros elementos, como mostram, na Figura 2.8, as declarações dos elementos pessoa, codigo, nome e endereco.

Ao declarar elementos, podem-se definir controles de conteúdo, de seqüência e de cardinalidade. Os controles de conteúdo, segundo ANDERSON *et al.* (2000), classificam-se em quatro tipos:

- empty: um elemento é declarado do tipo EMPTY quando houver necessidade do elemento não possuir valor. Por exemplo: <!ELEMENT email EMPTY>
- element: um elemento do tipo ELEMENT denota a situação na qual se deseja declarar elementos filhos de um determinado elemento. Por exemplo: <!ELEMENT pessoa (nome, endereco) >
- any: ao declarar um elemento do tipo ANY, pode-se usar qualquer valor em termos de conteúdo para ele. Por exemplo: <!ELEMENT anamnese ANY >
- mixed: o conteúdo do tipo MIXED é uma combinação de conteúdo de elementos e dados caracteres, como por exemplo: <!ELEMENT secao (#PCDATA | subsecao*) >

Existem também controles de seqüência, que são utilizados em elementos complexos, ou seja, em elementos formados por outros elementos. O controle dá-se através de componentes do padrão DTD, chamados “conectores”, os quais indicam a seqüência dos elementos-filhos e dos conteúdos, além de definir opção entre elementos e conteúdos. Os controles de seqüência são:

- vírgula (,): especifica que o elemento antes da vírgula deve ser apresentado primeiro e tem de ser seguido pelo elemento após a vírgula;

- barra vertical (|): especifica uma condição OU, define que apenas um elemento ou dado caracter de um lado da barra vertical ou de outro lado da barra vertical deverá aparecer.

Outro tipo de controle é o controle de cardinalidade[4], que oferece mecanismos para controlar a freqüência com que os elementos em uma lista de filhos podem aparecer. Indicadores de ocorrência são os caracteres responsáveis por essa indicação. Podem ser:

- sinal de mais (+): um elemento seguido deste sinal indica que pode aparecer uma ou várias vezes (cardinalidade um para muitos). Um exemplo é a declaração do elemento PESSOA na Figura 2.8;
- asterisco (*): o elemento seguido deste sinal significa que ele pode aparecer zero ou várias vezes (cardinalidade zero para muitos). Um exemplo é a declaração do elemento CÓDIGO na Figura 2.8;
- ponto de interrogação (?): o elemento seguido deste sinal indica que ele pode não aparecer ou apenas aparecer uma vez (cardinalidade zero para um). Por exemplo, na Figura 2.8, a declaração do elemento NOME.

Um elemento que não possui indicador de cardinalidade significa que ele deve aparecer obrigatoriamente uma única vez (cardinalidade um para um). Além da declaração de elementos, em uma DTD podem-se declarar atributos. Para declarar um atributo deve-se utilizar a palavra-chave <!ATTLIST, seguida do nome do elemento que ele está especificando, o nome do atributo, o tipo de informação que irá conter e, por último, a declaração do padrão da declaração. Por exemplo, <! ATTLIST codigo id CDATA #REQUIRED >.

No exemplo, a palavra codigo indica o nome do elemento que irá conter o atributo; a palavra id identifica o nome do atributo; a palavra CDATA especifica o tipo de informação que o atributo poderá conter e #REQUIRED indica o padrão do atributo. A Tabela 2.2 mostra a classificação dos tipos de atributos com significados e exemplos.

[4] Cardinalidade é a quantidade de ocorrências que o elemento pode ter num documento XML.

Tabela 2.2- Tipos de atributos.

Tipo	**Significado**
CDATA	Define um atributo do tipo texto (*string*). Ex: <! ATTLIST pessoa codigo CDATA #IMPLIED >
ID	O atributo é um nome único dentro do documento. Ex: <! ATTLIST pessoa cpf ID #REQUIRED >
IDREF	O atributo é uma referência a um atributo do tipo ID com o mesmo valor. Ex: <! ATTLIST pessoa cidade IDREF #IMPLIED >
IDREFS	Referência a uma série de Ids, separados por espaços. Ex: <! ATTLIST pessoa cidades IDREFS #IMPLIED >
ENTITY	Referência a uma entidade externa predefinida. Ex: <! ATTLIST pessoa foto ENTITY #IMPLIED >
ENTITIES	Referência a uma série de entidades, separados por espaços. Ex: <! ATTLIST pessoa fotos ENTITIES #IMPLIED >
NMTOKEN	O valor do atributo tem que ser um nome XML formal, ou seja, precisa começar com uma letra ou sublinhado e as seguintes podem incluir dígitos, letras e sublinhados. O valor consiste de uma única palavra sem espaços. Ex: <! ATTLIST cidade uf NMTOKEN #REQUIRED >
NMTOKENS	São atributos que possuem valores compostos de vários *tokens*, cada um deve ser um *token* válido, e são separados por espaços. EX: <! ATTLIST pessoa senhas NMTOKENS #REQUIRED >
NOTATION	Usados para referenciar dados externos ligados ao documento XML ou definir um valor para o atributo. Ex: <! ATTLIST jpg SYSTEM “jpgviewer.exe” > <! ATTLIST imagem tipoimagem NOTATION (gif | jpg) “gif” >
[valores definidos]	Define um conjunto de valores que podem ser definidos para o atributo. Ex: <! ATTLIST pessoa ativo (sim | não) >

Fonte: Primária.

A declaração do padrão do atributo define a maneira como aparece no documento e pode conter três tipos de valores: #REQUIRED, #IMPLIED e #FIXED. A Tabela 2.3 mostra cada uma das declarações de padrões e seus respectivos significados.

Tabela 2.3- Padrões de atributos.

Padrões	**Significado**
# REQUIRED	É obrigatória a presença do atributo em toda instância do elemento.
# IMPLIED	O atributo pode opcionalmente aparecer.
# FIXED	O valor é fixo, não pode ser modificado.

Fonte: Primária.

XSD

DTDs definem a estrutura de documentos XML e ainda são utilizadas. Porém, com o uso da tecnologia XML algumas limitações de DTDs vêm sendo evidenciadas:

- dificuldade de escrita e entendimento: DTDs são escritas em EBNF (*Extended Backus Naur Form*), que é uma linguagem não XML;

- dificuldade de processamento: os *parsers* de validação necessitam carregar e ler a DTD antes de validar conforme um determinado documento e não podem ser manipuladas através da linguagem DOM[5];
- não são extensíveis: cada vez que se deseja incluir algum elemento na DTD é necessário escrevê-la novamente. Não se pode referenciar algo sem criar uma entidade externa;
- ao utilizar *namespaces*[6], todas as regras devem existir na DTD;
- oferecem poucos tipos de dados;
- não suportam herança.

Devido a essas dificuldades, o W3C nomeou um comitê para trabalhar no problema resultando na especificação do XSD (XML *Schema Definition*). Pode-se referir a um XSD como um metadado, ou seja, dados sobre dados que expressam vocabulários compartilhados que permitem definir a estrutura, o conteúdo e a semântica de documentos XML, conforme um modelo definido num padrão XML *Schema* (ANDERSON *et al.*, 2000; W3C; BRADLEY ,2004).

Um esquema XSD, conforme mostra a Figura 2.9, consiste num elemento raiz denominado schema, composto por uma variedade de subelementos, que podem ser do tipo simples (simpleType) ou do tipo complexo (complexType), os quais determinam a aparência dos elementos e seus conteúdos na instância do documento.

Os elementos podem ter um prefixo xsd, o qual está associado com um *namespace* declarado no elemento schema, conforme mostra a Figura 2.9. O propósito dessa associação é identificar os elementos e os tipos simples pertencentes a um determinado vocabulário.

Segundo BRADLEY (2004), "o principal propósito de uma definição de XSD é declarar os elementos e atributos que constituem um documento modelo". Para declarar novos tipos de elementos utiliza-se a palavra element, seguida da palavra name, que identifica o nome do elemento em questão, além de poder declarar o tipo de dado que

[5] DOM é a interface de programação padronizada pelo W3C para tratar documentos XML.
[6] *Namespaces* são uma coleção de nomes identificados por uma URI (*Uniform Resource Identifier*). São utilizados como solução para problemas de ambigüidade e colisão de nomes.

irá conter, declaração feita através da palavra type. Por exemplo, na Figura 2.9, a declaração do elemento PRIMEIRO.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="nome">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="primeiro" minOccurs="1" maxOccurs="1"/>
                <xs:element ref="meio" minOccurs="0" maxOccurs="1"/>
                <xs:element ref="sobrenome" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
            <xs:attribute name="codigo" type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="identificacao">
        <xs:complexType>
            <xs:choice>
                <xs:element ref="cgc"/>
                <xs:sequence>
                    <xs:element ref="rg"/>
                    <xs:element ref="cpf"/>
                </xs:sequence>
            </xs:choice>
        </xs:complexType>
    </xs:element>
    <xs:element name="pessoas">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="nome"/>
                <xs:element ref="identificacao"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="primeiro" type="xs:string"/>
    <xs:element name="meio" type="xs:string"/>
    <xs:element name="sobrenome" type="xs:string"/>
</xs:schema>
```

Fonte: Primária.

Figura 2.9- Exemplo de XSD.

Similarmente à criação de novos elementos, novos atributos podem ser declarados. A sua declaração é feita através da palavra attribute, seguida da palavra name, que dá o nome para referenciar o atributo. Por exemplo, a declaração do atributo CODIGO na Figura 2.9.

O conteúdo de um elemento pode ser simples ou complexo. Os elementos simples, ao contrário dos complexos, não possuem na sua estrutura especificação de elementos filhos nem podem carregar atributos. BRADLEY (2004) classifica os conteúdos de elementos da seguinte forma:

- qualquer conteúdo: neste caso qualquer conteúdo pode ser atribuído ao elemento. As declarações podem ser feitas das seguintes formas: <element name=”observacao” />, ou <element name=“observacao” type=“anyType” />.
- elementos vazios: geralmente fornecem instruções que criam ou referenciam outras declarações de elementos dentro do elemento corrente. Por exemplo, a Figura 2.10 mostra a declaração de um elemento vazio.

```
<element name = “comentario”>
    <complexType> </complexType>
</element>
```

Fonte: Primária.

Figura 2.10- Exemplo de elemento com conteúdo vazio.

- conteúdo simples: elementos com conteúdo simples ocorrem quando o estado do seu conteúdo está de acordo com um tipo de dado. O atributo type é adicionado ao elemento para esse propósito. Por exemplo, a declaração do elemento PRIMEIRO na Figura 2.9.
- elementos com valores padrão: valores padrão para elementos são aplicados apenas quando o elemento está presente, mas não contém conteúdo. O atributo default, adicionado na declaração do elemento, indica o valor. Por exemplo, <element name = “uf” type = “string” default = “RS” />;
- valores fixos: neste caso, é possível especificar um valor para o conteúdo de um elemento, que não pode ser modificado em qualquer instância do documento que o inclui. O atributo fixed, adicionado na declaração do elemento, indica o valor que será sempre aplicado. Por exemplo, <element name = “RS” type = “string” fixed = “Rio Grande do Sul” />;
- enumeração de valores possíveis: pode-se restringir o conteúdo de um elemento através de uma lista de valores possíveis. Por exemplo, a Figura 2.11 mostra uma enumeração de valores possíveis para o elemento uf.

```
<element name=”uf”>
   <simpleType>
      <restriction base = “string”>
         <enumeration value = “RS” />
         <enumeration value = “SC” />
      </restriction>
   </simpleType>
</element>
```

Fonte: Primária.

Figura 2.11- Exemplo de elemento com enumeração de valores possíveis.

- conteúdo misto: o termo “conteúdo misto” implica uma mistura de elementos e texto. Nesse caso, o elemento não possui elementos filhos. Por exemplo, <comunicado> Prezado Sr(a) <primeiro> Alexandre </primeiro>.

Outra maneira de se classificar o conteúdo de um elemento é através da utilização de elementos específicos, que informam exatamente o que é permitido dentro de cada elemento. Esses elementos podem ser:

- elementos filhos referência: neste caso, o atributo ref junto ao elemento element indica o nome do elemento o qual ele referencia, podendo, eventualmente, referenciar-se ao elemento pai, criando um *loop* infinito. O valor do elemento será o mesmo especificado no atributo name do elemento referenciado. Por exemplo, na Figura 2.9 a declaração <xs:element ref=”primeiro”/>.

- elementos tipo complexo: quando numa declaração de elementos é necessário especificar quais os elementos filhos são permitidos, ela deve conter um elemento complexType. Os elementos podem aparecer numa ordem predefinida, não ter restrições de ordem, ou consistir numa série de alternativas. Por exemplo, a declaração do elemento NOME na Figura 2.9.

  Também se podem realizar derivações a partir de um tipo complexo já existente. O elemento complexContent especifica que um tipo complexo é derivado de outro tipo complexo; pode conter os elementos extension e restriction, que indicam, respectivamente, uma derivação por extensão e uma derivação por restrição.

- elementos seqüência: quando elementos filhos necessitam ocorrer numa determinada ordem na instância de um documento, utiliza-se o elemento sequence. Por exemplo, na Figura 2.9 a declaração do elemento NOME.

- elementos opcionais e repetitivos: os atributos minOccurs e maxOccurs são adicionados ao elemento para indicar, respectivamente, o mínimo e máximo de vezes que ele pode ocorrer. Por exemplo, na Figura 2.9 as declarações dos elementos referências PRIMEIRO, MEIO, SOBRENOME.

- elementos de escolha: elementos podem ser opcionais; no caso de dois elementos, pode se referir a apenas um na instância do documento, utilizando-se, para isso, o elemento choice. Por exemplo, na Figura 2.9 a declaração do elemento identificacao.

- elemento all: através desse elemento pode-se definir que todos os elementos devem estar presentes, mas não necessariamente na mesma ordem em que são referenciados. A Figura 2.12 mostra um exemplo de definição do elemento all.

```
<all>
   <element ref="CPF" />
   <element ref="RG" />
   <element ref="PIS" minOccurs="0"/>
</all>
```

Fonte: Primária.

Figura 2.12- Exemplo de elemento all.

- modelos complexos: os elementos choice e sequence podem estar embutidos um dentro do outro qualquer número de vezes e em qualquer número de níveis, também dentro de outra instância do mesmo elemento. Na Figura 2.9 a declaração de elemento identificacao é um exemplo de modelo complexo.

- elemento any: às vezes, nas instâncias do documento, necessita-se de mais liberdade. Uma opção é utilizar o elemento any no documento.Um exemplo é mostrado na Figura 2.13.

```
<element name="fotos">
   <complexType>
      <sequence>
         <any minOccurs="0"
              maxOccurs="unbouded"
              namespace="##local"
         />
      </sequence>
   </complexType>
</element>
```

Fonte: Primária.

Figura 2.13 - Exemplo do elemento any.

- grupos compartilhados: quando dois ou mais elementos possuem conteúdos de estruturas idênticas, não necessitam serem replicados. O elemento group cria um componente reusável com conteúdo completo ou parcial do modelo de que se

necessita. Esse grupo é referenciado através de um nome. Por exemplo, a Figura 2.14 mostra a definição de um grupo compartilhado.

```
<schema>
   <group name="identificacao">
      <sequence>
         <element ref="nome" />
         <element ref="endereco" />
      </sequence>
   </group> ...
</schema>
```

Fonte: Primária.

Figura 2.14- Exemplo do elemento group.

A declaração de atributos, similarmente à declaração de elementos, pode conter especificação do tipo de conteúdo. A Figura 2.15 mostra alguns exemplos dessas declarações.

Segundo BRADLEY (2004), existem várias maneiras de anexar atributos a uma declaração de elementos. Podem-se ter atributos em elementos vazios, nesse caso, o elemento é considerado complexo, como mostra na Figura 2.15 a declaração do atributo PRIMEIRO no elemento NOME. Outra maneira de restringir o conteúdo dos atributos é associá-lo a um tipo de dado simples. Por exemplo, <foto formato="JPG" referencia="c:\fotos\foto01.jpg" />.

Podem-se declarar atributos em elementos com conteúdo; quando o elemento possui conteúdo e atributos, não basta simplesmente colocar atributo type na declaração do elemento visando restringir o seu conteúdo. Existem duas maneiras de definir este caso. A primeira utiliza o conceito de conteúdo misto e, portanto, requer um elemento complexType e um atributo mixed com um valor true atribuído. Um exemplo é mostrado na Figura 2.15, com a declaração do atributo complemento no elemento endereco.

A segunda alternativa explora o fato de que um elemento é complexo pelo fato de conter atributos, não por conter elementos filhos. É necessário o elemento complexType, porém, para declarar os atributos, são incluídos um elemento simpleContent e um elemento extension, com o atributo base anexado, o qual referencia o tipo de dados que será estendido. Por exemplo, na Figura 2.15 a declaração do atributo UF no elemento CIDADE. A existência de um atributo na instância de um

documento é, por padrão, opcional, entretanto a sua presença pode ser obrigatória ou proibida. O atributo use é utilizado para especificar essas declarações, podendo conter um dos valores: optional, required e prohibited. Por exemplo, <attribute name = “id” use = “required” ... />.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="nome">
        <xs:complexType>
            <xs:attribute name="primeiro" type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="endereco">
        <xs:complexType mixed="true">
            <xs:attribute name="complemento" type="xs:string"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="cidade">
        <xs:complexType>
            <xs:simpleContent>
                <xs:extension base="xs:string">
                    <xs:attribute name="uf" type="estados" default=”SC”/>
                </xs:extension>
            </xs:simpleContent>
        </xs:complexType>
    </xs:element>
    <xs:simpleType name="estados">
        <xs:restriction base="xs:string">
            <xs:enumeration value="RS"/>
            <xs:enumeration value="SC"/>
        </xs:restriction>
    </xs:simpleType>
    <xs:element name="pessoas">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="nome"/>
                <xs:element ref="endereco"/>
                <xs:element ref="cidade"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Fonte: Primária.

Figura 2.15- Exemplos de declarações de atributos.

Um atributo pode ter um valor padrão (*default*), que se aplica apenas quando o atributo está ausente. Para definir o valor padrão de um atributo, utiliza-se o atributo default. Por exemplo, na Figura 2.15 a declaração do atributo UF.

Também é possível especificar um valor para um atributo que não pode ser modificado em uma instância do documento. Através do atributo fixed é especificado o valor que será aplicado. Por exemplo, <attribute name = “SC” type = “string” fixed =

“Santa Catarina”/>. Pode-se também restringir o conteúdo de atributos através da enumeração dos valores possíveis para ele. Por exemplo, na Figura 2.15 a declaração do tipo de dados ESTADOS e a sua referência na declaração do atributo UF.

Outra maneira de impor restrições ao conteúdo de um elemento ou atributo específico é através da definição do seu tipo de dado. Tipos de dados são geralmente baseados em outros tipos de dados similares, porém menos restritivos, assim formando uma hierarquia de tipos.

Se necessário, podem-se criar novos tipos de dados simples, que são derivados de outros tipos de dados simples já existentes, através do uso elemento simpleType. Pode-se usar este elemento também na definição de elementos (element) e atributos (attribute), caso eles não utilizem os atributos ref e type. Por exemplo, a Figura 2.15 mostra a utilização de um elemento simpleType na declaração do tipo ESTADOS. Os tipos simples podem ser compartilhados e referenciados por um *namespace*, desde que sejam declarados no elemento *schema*.

Um tipo de dados restrito é definido através de especificações de restrições aplicadas a um tipo de dados existente, conhecidas como *facets*. A Tabela 2.4 mostra os tipos de *facets* e as suas definições e o Anexo 1 mostra uma associação dos tipos de *facets* permitidos com os tipos simples de dados.

Tabela 2.4 - Tipos de *facets*.

Facets	**Definição**
Length	Restringe o valor para um número fixo de caracteres.
Minimum length	Restringe o valor para um número mínimo de caracteres.
Maximum length	Restringe o valor para um número máximo de caracteres.
Pattern	Define um template contra o qual o valor deve ser comparado.
Enumeration	Define as possibilidades de valores.
Whitespace	Define a maneira que os caracteres de espaço (alimentação de linha, espaço em branco, tabulação, ..) aparecem. Pode ter os valores “replace”, “collapse” e “preserve”.

Fonte: Primária.

Tipos simples que possuem valores numéricos têm *facets* adicionais que são chamadas “*facets* enumeradas”, as quais especificam a gama de valores possíveis para valores mínimo e máximo e, também, o total máximo de dígitos e a parte fracionária de um valor. A Tabela 2.5 mostra os tipos de *facets* enumeradas e as suas definições e o Anexo 2, uma associação dos tipos de *facets* enumeradas e os tipos simples de dados.

Para representar cada tipo de *facets* são utilizados elementos distintos. Os elementos utilizados são: length, minLength, maxLength, pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, maxInclusive, totalDigits e fractionDigits. O Anexo 3 mostra esses elementos com exemplos e comentários.

Tabela 2.5- Tipos de *facets* enumeradas.

Facets	**Definição**
Maximum inclusive	Especifica o valor máximo, incluindo o próprio valor.
Maximum exclusive	Especifica o valor máximo, excluindo o próprio valor.
Minimum inclusive	Especifica o valor mínimo, incluindo o próprio valor.
Minimum exclusive	Especifica o valor mínimo, excluindo o próprio valor.
total digits	Especifica o número máximo de dígitos permitidos no valor numérico.
Fractional digits	Especifica o número de dígitos que seguem o ponto decimal.

Fonte: Primária.

Também é possível validar se cada ocorrência de um tipo de elemento, dentro de uma instância do documento, contém valores diferentes de outras ocorrências do mesmo elemento e checar se uma referência de um elemento a outro é válida.

Fazendo uma analogia com o banco de dados relacional, cada linha de uma tabela consiste de vários campos, que devem ser identificados unicamente através de um valor de um campo, ou de uma combinação de valores de vários campos; valor ou valores desses campos formam a chave (*key*), que identifica unicamente a linha. Em XML, pode-se dizer que fragmentos[7] são equivalentes a linhas e que os elementos e atributos são equivalentes aos campos; portanto, a chave deve ser o conteúdo de um elemento ou, talvez, a combinação de valores de vários atributos.

Se uma linha, numa tabela no banco de dados relacional, envolve vários relacionamentos com linhas de outras tabelas, tem-se uma referência. Num documento XML, uma referência é feita através da colocação de uma cópia da chave no elemento ou atributo referência.

Uma definição de esquema pode conter informações que identificam fragmentos que devem conter valores únicos. O elemento unique é inserido na declaração de elementos que irão conter os fragmentos. O atributo name dá um nome que identifica o

[7] Um fragmento é usado para descrever qualquer elemento que possui existência própria e que contém todas as informações necessárias para ser identificado ou localizado.

fragmento único. A Figura 2.16 (a) mostra um exemplo de definição do elemento unique.

```
<schema ...>
    <xs:element name="pessoa">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="nome" maxOccurs="unbounded"/>
                <xs:element ref="cursos" maxOccurs="unbounded" />
            </xs:sequence>
        </xs:complexType>
        <xs:unique name="idcodigo">
            <xs:selector xpath="nome"/>
            <xs:field xpath="codigo"/>
        </xs:unique>
        <xs:keyref name="refcodigo" refer="idcodigo">
            <xs:selector xpath="cursos"/>
            <xs:field xpath="codigopessoa"/>
        </xs:keyref>
    </xs:element>
    <xs:element name="nome">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="codigo" minOccurs="1" maxOccurs="1"/>
                <xs:element ref="primeiro" minOccurs="1" maxOccurs="1"/>
                <xs:element ref="sobrenome" maxOccurs="unbounded"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="cursos">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="codigopessoa" minOccurs="0" maxOccurs="1"/>
                <xs:element ref="nomecurso" minOccurs="0" maxOccurs="1"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="codigo" type="xs:integer"/>
    <xs:element name="primeiro" type="xs:string"/>
    <xs:element name="sobrenome" type="xs:string"/>
    <xs:element name="codigopessoa" type="xs:integer"/>
    <xs:element name="nomecurso" type="xs:string"/>
</schema>
```

(a)

```
<xs:key name="idcodigo">
    <xs:selector xpath="nome"/>
    <xs:field xpath="codigo"/>
</xs:key>
```

(b)

Fonte: Primária.

Figura 2.16 - Exemplo de referências de chaves.

O elemento key é quase idêntico ao elemento unique, possuindo o mesmo conteúdo de opções e atributos, inclusive o atributo name. A diferença é que o elemento key não permite valores nulos ou vazios para o identificador, ao passo que o elemento unique permite. A Figura 2.16 (b) mostra um exemplo onde o elemento unique na Figura 2.16 (a) pode ser substituído pelo elemento key.

Ao utilizar os elementos unique ou key, duas tarefas devem ser realizadas: na primeira, os fragmentos devem ser identificados, por exemplo, o nome de cada pessoa; na segunda, o valor, ou valores, desses fragmentos que servirão como chave deverá ser isolado. Por exemplo, o valor do CPF de cada nome de pessoa (BRADLEY, 2004).

O elemento selector realiza a tarefa de encontrar fragmentos. O atributo xpath, que identifica os fragmentos através de sua localização, é requerido no elemento selector e indica o caminho através de uma expressão *XPath*[8].

Os valores do atributo xpath devem estar conformes às restrições do padrão da linguagem *XPath*. A Tabela 2.6 mostra os possíveis valores e seu significado.

Tabela 2.6 - Expressões *XPath*.

Expressão Xpath	**Significado**
*	Todos elementos filhos.
A	Elementos filhos de A.
child::A	Elementos filhos de A.
A/B	Elementos filhos B dos elementos filhos A.
A | B	Elementos filho A e elementos filho B.
A/B | C/D	Elementos filhos B dos elementos filhos A e Elementos filhos D dos elementos filhos C.
. / / *	Todos elementos descendentes.
. / / B	Todos elementos descendentes de B.
A / / B	Todos elementos descendentes de B com todos elementos filho de A.

Fonte: Primária.

O elemento field é usado nos elementos unique e key para encontrar os valores, os quais serão chaves dentro de um fragmento. Pelo menos um elemento field deve estar presente logo após a definição do elemento selector. O atributo xpath também é utilizado e o uso do símbolo "@" antes do nome indica um nome de atributo, não o nome de um elemento.

A maneira como referências em um documento XML são identificadas é igual à maneira como chaves são identificadas. A localização de um fragmento contendo a referência e a localização do campo ou campos que constituem o valor de referência é novamente encontrada através dos elementos selector e field, respectivamente. Esses elementos são colocados dentro de um elemento keyref.

[8] Linguagem de consulta a documentos XML.

O atributo name deve ser adicionado ao elemento keyref para dar o nome à referência. Uma referência deve identificar um conjunto específico de chaves, representadas pelos elementos unique ou key. O valor do atributo refer indica o nome do conjunto de chaves a que será feita a referência. Os campos de referência devem, necessariamente, ser do mesmo tipo de dados. A Figura 2.16 (a) mostra um exemplo de definição de um XSD utilizando o elemento keyref.

2.3 Considerações finais

Neste capítulo foi realizado, primeiramente, um estudo das restrições de integridade do modelo relacional. A escolha deste modelo justifica-se por apresentar uma teoria consolidada para o tratamento de restrições de integridade em bancos de dados relacionais.

Através desse estudo, verificou-se por meio da análise das obras de diversos autores que existem várias classificações de restrições de integridade e estabeleceu-se uma nova classificação de restrições de integridade abrangendo essas obras

Na seção 2.2.2 foi realizado um estudo dos recursos que a linguagem SQL disponibiliza para tratamento de restrições de integridade, pois a SQL é a linguagem padrão para manipulação de bancos de dados relacionais. Além disso, constatou-se que, na categoria de restrições de integridade de domínio, as restrições são impostas pela linguagem SQL através da definição das entidades, especificando os tipos de dados dos atributos, através de gatilhos, de procedimentos, de cláusulas *checks* e asserções.

Por fim, tem-se um estudo da tecnologia XML, no qual, além de mostrar o crescimento da utilização e a importância de documentos XML para troca de dados na *Web*, objetivou-se levantar que controles existem nas especificações de esquemas, DTD e XSD, para imposição de restrições de integridade em documentos XML.

Através desse estudo constatou-se que a especificação do XSD é mais completa com relação à especificação da DTD, pois dispõe de mais recursos para impor controles em documentos XML.

As principais vantagens do XSD com relação a DTD são salientadas principalmente no tratamento das restrições de integridade de domínio, em específico restrições de atributo, que são impostas através do uso de *facets*. Outra vantagem é o tratamento de restrições de chaves e de integridade referencial, as quais são feitas através dos elementos *key*, *unique* e *keyref*.

3 RESTRIÇÕES DE INTEGRIDADE PARA DADOS XML

Este capítulo, primeiramente, mostra três estudos comparativos referentes às áreas relacionadas a esta dissertação, vistas no capítulo 2, com o objetivo de analisar quais controles a linguagem SQL, a DTD e o XSD possuem para impor a classificação de restrições de integridade estabelecida.

Em seguida, com base nas carências mostradas nos estudos comparativos, é definido o foco da dissertação e são apresentados alguns trabalhos relacionados existentes na literatura que objetivam suprir essas carências.

Por fim, com base em algumas deficiências dos trabalhos relacionados, é estabelecida a motivação desta dissertação.

3.1 Estudos comparativos

Nesta seção são mostrados três importantes estudos comparativos relacionados ao capítulo 2. O primeiro estudo mostra um comparativo entre a classificação de restrições de integridade do modelo relacional estabelecida e o seu tratamento pela linguagem SQL. O segundo e o terceiro estudos realizam um comparativo da classificação de restrições de integridade do modelo relacional e o suporte dado pela DTD e o XSD, respectivamente.

O objetivo destes estudos é verificar quais controles existem na linguagem SQL, na especificação da DTD e na especificação do XSD e se atendem a todas as categorias de restrições de integridade do modelo relacional estabelecidas na nova classificação.

3.1.1 Consideração de restrições de integridade relacionais por SQL

Um aspecto importante no estudo é saber de que maneira a teoria de restrições de integridade em bancos de dados relacionais é imposta na prática através da linguagem SQL. A Tabela 3.1 mostra este comparativo.

Tabela 3.1- Comparativo relacional vs. SQL.

SQL / **Relacional**	**Domínio**	**Básicas de Tabelas**	**Gerais**	**Gatilhos**	**Procedi-Mentos**	**Transa-ções**	**Momento de verificação**
a) Domínio							
Atributos	*domain, check*	*check*	—	*trigger*	*stored procedure*	—	*not deferrable ou deferrable*
Tipo	—	tipos de dados	—	—	—	—	*not deferrable*
Tuplas	—	—	*assertion, check*	*trigger*	*stored procedure*	—	*not deferrable ou deferrable*
Banco de dados	—	—	*assertion, check*	*trigger*	*stored procedure*		*not deferrable ou deferrable*
Transição de estados	*domain, check*	—	—	*trigger*	*stored procedure*	—	*not deferrable ou deferrable*
b) Chaves							
candidatas	—	*unique,*	—	—	—	—	*not deferrable ou deferrable*
primárias	—	*primary key*	—	—	—	—	*not deferrable ou deferrable*
estrangeiras	—	*foreign key*	—	—	—	—	*not deferrable ou deferrable*
c) Integridade referencial	—	*Primary key e foreign key*	—	*trigger*	*stored procedure*	—	*not deferrable ou deferrable*
d) Momento de Verificação	Imediato ou postergado	Imediato ou postergado	Imediato ou postergado	Imediato ou postergado	Imediato ou postergado	Imediato ou postergado	*not deferrable ou deferrable*
e) Baseada em eventos	—	—	—	*trigger*	*stored procedure*	—	*not deferrable ou deferrable*

Fonte: Primária.

Na Tabela 3.1, a linha X mostra a classificação das categorias de instruções existentes na linguagem SQL para a imposição de restrições de integridade. A coluna Y mostra a classificação das categorias estabelecidas para o modelo relacional. Na

intersecção entre X e Y têm-se as instruções SQL correspondentes a sua categoria no eixo X e no eixo Y, e o caracter "-" indica a inexistência de instruções.

Analisando a Tabela 3.1, verifica-se que as restrições de domínio do modelo relacional são na sua totalidade atendidas de alguma maneira pela linguagem SQL. Com exceção das restrições de tipo, impostas através da definição do tipo de dado, as demais restrições de domínio são, sobretudo, impostas através do uso da cláusula CHECK, gatilhos e procedimentos. Com relação ao momento de verificação, nota-se que, exceto as restrições de tipo, que são do tipo imediato, as demais categorias as restrições de integridade podem ser de verificação imediata ou postergada.

Também se verifica que as restrições de chaves estão associadas à definição de tabelas através das cláusulas PRIMARY KEY, FOREIGN KEY e UNIQUE. Essas cláusulas também são utilizadas para impor restrições de integridade referencial, porém existe a possibilidade de impor essas restrições com o uso de recursos como procedimentos e gatilhos.

As restrições quanto ao momento de verificação são todas do tipo imediato ou do tipo postergado. Já as restrições baseadas em eventos podem ser declaradas na linguagem SQL através do uso de procedimentos (chamados a partir de uma aplicação) e também por gatilhos (disparados automaticamente pelo sistema de gerência do banco de dados).

Portanto, com base na análise da Tabela 3.1, pode-se concluir que a linguagem SQL, através do seu grupo de instruções, atende a todas as categorias de restrições de integridade do modelo relacional.

3.1.2 Relacional vs. Esquemas XML

Outro enfoque importante do estudo é saber se as restrições de integridade do modelo relacional podem ser impostas em documentos XML e de que formas. Esta seção visa mostrar este estudo e, também, levantar lacunas de restrições de integridade não preenchidas pela especificação da tecnologia XML.

Um documento XML, no que se refere a restrições, pode ser classificado em *bem formado* e *válido*. Neste estudo, o enfoque são os documentos *válidos*, ou seja, documentos que devem estar de acordo com um esquema DTD ou XSD.

Dados XML, em razão de suas características (representação autodescritiva, estrutura irregular, mistura de texto e estrutura, ...), são classificados como dados semi-estruturados (ABITEBOUL *et al*., 2000). Por esse fato, o comparativo de esquemas XML com as classificações de restrições de integridade do modelo relacional mostra que XML necessita de adaptações referentes aos conceitos das categorias de restrição de integridade, como por exemplo, a categoria de domínio mostrada a seguir.

Uma restrição de integridade de tipo, no contexto XML, abrange restrições de estrutura, como cardinalidade, seqüência e escolha. Já a restrição de integridade de atributo, no contexto XML, abrange os valores válidos para os elementos ou atributos no documento XML, incluindo as restrições de transição de estados. A categoria de restrição de integridade de tuplas abrange a validação dos elementos ou atributos pertencentes ao mesmo nível hierárquico do documento XML e a categoria de banco de dados, abrange a validação de elementos ou atributos pertencentes a níveis hierárquicos diferentes do documento XML.

A seguir, são mostrados os estudos comparativos entre esquemas DTD e XSD com a classificação de restrições de integridade do modelo relacional estabelecida.

Relacional vs. DTD

Baseado no estudo feito no capítulo 2, uma DTD, no que se refere à imposição de restrições de integridade, possui controles de conteúdo de elementos, controles de seqüência e escolha de subelementos e controles de cardinalidade, além de poder especificar na declaração de atributos o seu tipo e a obrigatoriedade ou não de sua ocorrência. A Tabela 3.2 mostra uma associação desses controles com a classificação de restrições de integridade do modelo relacional.

Analisando a Tabela 3.2, verifica-se que o maior número dos controles para impor restrições que a DTD disponibiliza concentra-se na categoria de restrições de domínio

no modelo relacional. Os controles de conteúdos de elementos (ANY, EMPTY, MIXED, PCDATA) podem ser caracterizados como restrições de tipo, pois definem o tipo de conteúdo do elemento.

Tabela 3.2- Comparativo DTD vs. relacional.

DTD / Relacional	Conteúdos de elementos	Seqüência e escolha de subelementos	Cardinalidade de elementos	Tipos de atributos	Ocorrência de atributos
a) Domínio					
Tipo	ANY, EMPTY, MIXED, PCDATA	VÍRGULA, BARRA VERTICAL	+, *. ?	CDATA, ENTITY, ENTITIES, NMTOKEN, NMTOKENS,NOTATION	–
Atributos	–	–	–	–	REQUIRED, IMPLIED, FIXED
Tuplas	–	–	–	–	–
Banco de dados	–	–	–	–	–
b) Chaves					
Candidatas	–	–	–	ID	–
Primárias	–	–	–	ID	–
Estrangeiras	–	–	–	IDREF	–
c) Integridade referencial	–	–	–	ID, IDREF	–
d) Momento de Verificação	IMEDIATO	IMEDIATO	IMEDIATO	IMEDIATO	IMEDIATO
e) Baseada em eventos	API + *Parser*	API + *Parser*	API + *Parser*	API + *Parser*	API + *Parser*

Fonte: Primária.

Os controles de seqüência (VÍRGULA, BARRA VERTICAL) também podem ser classificados como uma restrição de tipo, pois são utilizados para determinar a estrutura dos elementos, através da seqüência do conteúdo dos elementos-filhos, além de definir a opção entre elementos e conteúdos do tipo texto.

Os controles de cardinalidade (*, +, ?) também podem ser enquadrados como uma restrição de tipo, pois descrevem quais são os elementos que os compõem e restringem as suas quantidades de ocorrência, ou seja, define a sua estrutura.

Os controles de tipos de atributos (CDATA, ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS, NOTATION) e seus valores permitidos podem ser classificados em duas categorias no modelo relacional: restrição de domínio, mais especificamente de tipo, pois definem o tipo de dado do atributo, e restrição de chaves, através das cláusulas ID e IDREF.

Os padrões de ocorrência de atributos (REQUIRED, IMPLIED, FIXED) podem ser classificados como uma restrição de atributo, pois irão definir a obrigatoriedade, a opcionalidade ou um determinado valor fixo para o atributo.

Os controles de restrições baseadas em eventos podem ser obtidos através do uso das APIs (*Application Programming Interface*) de manipulação de estruturas XML, SAX[9] (*Simple API for XML*) e DOM[10] (*Document Object Model*), com uma linguagem de programação que oferece suporte para as APIs, gerando um *parser* (código de verificação). Porém, esses recursos não fazem parte da especificação da DTD.

Quanto ao momento de verificação, as restrições são verificadas sempre no momento em que o *parser* é executado, ou seja, são sempre imediatas (neste caso, assume-se que o *parser* é sempre executado).

Os controles de integridade referencial são menos específicos do que os utilizados em bancos de dados relacionais. Os tipos de atributos ID e IDREF são utilizados para impor este tipo de restrição, porém um atributo IDREF pode referenciar qualquer elemento que possua um atributo ID no documento XML.

Pela Tabela 3.2 nota-se, na especificação da DTD, a ausência de controles para imposição de restrições de integridade de tuplas e de banco de dados semelhantes às existentes no modelo relacional. Ainda, o controle de restrições de integridade de atributo é considerado parcial, pois não existem controles que possam validar os possíveis valores de um elemento ou atributo durante uma transição de estados.

Relacional vs. XSD

[9] SAX é uma API dirigida a eventos para processamento de dados XML. (SILVA, 2004)
[10] DOM é uma API orientada a um modelo hierárquico para acesso a componentes de um documento XML. (SILVA, 2004)

Assim como na DTD, o XSD também possui controles que podem ser utilizados para impor restrições em documentos XML, porém eles são mais amplos, tanto em quantidade como em qualidade, que os existentes na DTD, ou seja, possibilitam o tratamento de um maior número de restrições de integridade.

Esses controles podem ser classificados como conteúdo de elementos, seqüência e escolha de elementos, padrões de elementos, cardinalidade de elementos, conteúdo de atributos, padrões de atributos, ocorrência de atributos, tipos de dados e chaves e referências. A Tabela 3.3 mostra um comparativo dos recursos existentes no XSD, semelhantes aos existentes na DTD, com as categorias de restrições do modelo relacional. A Tabela 3.4 mostra um comparativo dos recursos existentes no XSD, adicionais aos existentes na DTD, com as categorias de restrições do modelo relacional.

Tabela 3.3- Comparativo XSD vs. relacional (I).

XSD / **Relacional**	**Conteúdo de elementos**	**Seqüência e escolha de subelementos**	**Cardinalidade de elementos**	**Tipos de dados de atributos e elementos**	**Ocorrência de atributos**
a) Domínio					
Tipo	Qualquer conteúdo, Elementos vazios, Conteúdo simples, Conteúdo misto, Elementos referencia, Elementos complexos, Elementos ANY, Elementos GROUP	Elemento SEQUENCE, Elemento CHOICE, Modelo complexo (SEQUENCE e CHOICE)	Elementos opcionais repetitivos (maxOccurs e minOccurs), Elemento ALL	Tipos simples (SIMPLETYPE, FACETS), Tipos complexos (Derivações por extensão e restrição)	-
Atributos	_	_	_	_	Requeridos e opcionais (USE)
Tuplas	_	_	_	_	_
Banco de dados	_	_	_	_	_
b) Chaves					
Candidatas	_	_	_	_	_
Primárias	_	_	_	_	_
Estrangeiras	_	_	_	_	_
c) Integridade referencial	_	_	_	_	_
d) Momento de Verificação	IMEDIATO	IMEDIATO	IMEDIATO	IMEDIATO	IMEDIATO
e) Baseada em eventos	API + *Parser*	API + *Parser*	API + *Parser*	API + *Parser*	API + *Parser*

Fonte: Primária.

Analisando, de uma maneira geral, as Tabelas 3.3 e 3.4, nota-se que a maior parte dos controles do XSD se enquadra na categoria de restrição de domínio do modelo relacional, existindo principalmente controles para tratar restrições de tipo e restrições de atributos.

Tabela 3.4- Comparativo XSD vs. relacional (II).

XSD / Relacional	Conteúdo de atributos	Padrões de atributos	Padrões de elementos	Chaves e referências
a) Domínio				
Tipo	Em elementos vazios, Em elementos com conteúdo, Pertencentes a *namespaces*	–	–	–
Atributos	–	Com valores padrão (DEFAULT), Com valores fixos (FIXED), Valores enumerados (ENUMERATION)	Valores padrão (DEFAULT), Valores fixos (FIXED), Valores Enumerados (ENUMERATION)	–
Tuplas	–	–	–	–
Banco de dados	–	–	–	–
b) Chaves				
Candidatas	–	–	–	UNIQUE, KEY
Primárias	–	–	–	KEY
Estrangeiras	–	–	–	KEYREF
c) Integridade referencial	–	–	–	UNIQUE, KEY, KEYREF
d) Momento de Verificação	IMEDIATO	IMEDIATO	IMEDIATO	IMEDIATO
e) Baseada em eventos	API + *Parser*	API + *Parser*	API + *Parser*	API + *Parser*

Fonte: Primária.

Os controles de conteúdo de elementos, controles de seqüência e escolha de elementos, controles de cardinalidade de elementos, controles de conteúdo de atributos e controles de ocorrência de atributos são associados às restrições de tipo do modelo relacional, pois definem a estrutura dos elementos e dos atributos aos quais estão associados.

A especificação de tipos de dados também pode ser associada ao modelo relacional como uma restrição de tipo, pois permite definir restrições que fazem parte do tipo de dado que poderá ser declarado para um elemento e/ou atributo. Na coluna tipos de dados de elementos e atributos, na Tabela 3.3, incluem-se as características de restrições de tipos simples (*facets*) e derivação por restrição e extensão.

Os controles de padrões de elementos e controles de padrões de atributos são classificados no modelo relacional como restrições de atributos, por estarem definindo valores possíveis para um elemento ou atributo, o que pode ser visto na Tabela 3.4.

Na especificação XSD existem controles para tratamento de chaves e integridade referencial semelhantes ao modelo relacional, o que é feito através da declaração dos elementos UNIQUE, KEY e KEYREF, associados a outros elementos através de expressões *XPath*. As chaves candidatas são definidas através dos elementos UNIQUE ou KEY. As chaves primárias podem ser definidas através do elemento KEY, pois seu valor é obrigatório no documento XML, ao contrário do elemento UNIQUE, cujo valor não é obrigatório e que pode utilizado para definir chaves alternativas. O elemento KEYREF é utilizado para fazer referência a um elemento KEY ou UNIQUE, impondo, dessa forma, controles de integridade referencial.

Quanto ao momento de verificação, à semelhança da DTD, todas as verificações são também feitas no momento de execução do *parser*. Já as restrições baseadas em eventos podem ser impostas no documento XML também através do uso das APIs SAX e DOM com uma linguagem de programação que suporte essas APIs, gerando um *parser* que efetua a validação.

Através da análise das Tabelas 3.3 e 3.4 também se constata que, na especificação do XSD, não existem especificações para as categorias de restrições de integridade de tuplas e restrições de integridade de banco de dados semelhantes às existentes no modelo relacional. Também, da mesma forma que na DTD, não existem controles para a categoria de restrições de integridade de transição de estados.

Como mostrado, a especificação de esquemas XML não é expressiva suficiente para suportar todos os tipos de restrições de integridade de bancos de dados, mais especificamente, do modelo relacional.

O estudo mostrou que existem carências de recursos nas especificações da DTD e do XSD para a imposição de restrições de integridade, principalmente com relação às categorias de restrições de integridade de domínio e baseadas em eventos.

Portanto, o foco desta dissertação é o controle de restrições de integridade de domínio utilizando uma abordagem ECA (Evento-Condição-Ação), similar a recursos da linguagem SQL como gatilhos e *checks*, ou seja, toda vez que ocorre uma atualização de dados, verificações de integridade são realizadas.

Na seção 3.2 são mostrados alguns trabalhos que buscam atender às necessidades de imposição de restrições de integridade em documentos XML. Porém, esses trabalhos posssuem algumas limitações que esta dissertação visa suprir.

3.2 Trabalhos relacionados

Em virtude da necessidade de imposição de restrições de integridade em documentos XML, alguns pesquisadores, citados a seguir, têm concentrado esforços em trabalhos que buscam estender a tecnologia XML com a incorporação de restrições de integridade.

Fazendo uma analogia com o modelo relacional, podem-se concentrar os trabalhos em três grupos de restrições de integridade: restrições de domínio, restrições de chaves e integridade referencial e restrições baseadas em eventos. As restrições quanto ao momento de verificação não são citadas, pois são consideradas sempre do tipo IMEDIATO, no momento da execução do *parser*. A seguir são mostrados alguns trabalhos referentes a esses grupos de restrições, com detalhamento dos trabalhos referentes a restrições de integridade de domínio e restrições de integridade baseada em eventos, as quais são o foco desta dissertação.

OGBUJI (2001) apresenta uma linguagem de validação para documentos XML chamada Schematron. Esta linguagem, ao invés de declarar esquemas, define uma série de regras de validação que são aplicadas dentro de um XSD que o validam contra uma instância do documento XML, como mostra a Figura 3.1.

Schematron depende quase que exclusivamente de expressões *XPath* para definir suas regras de validação. Uma especificação Schematron consiste numa referência *namespace* no elemento schema do XSD, que referencia uma série de padrões de elementos usados no documento XSD para efetuar as validações. Um *parser* é utilizado posteriormente para realizar essas validações no documento XML.

```
<?xml version=”1.0”?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.demo.org" xmlns="http://www.demo.org"
      xmlns:sch="http://www.ascc.net/xml/schematron"
      elementFormDefault="qualified">
   <xsd:annotation>
      <xsd:appinfo>
         <sch:title>Schematron validation</sch:title>
         <sch:ns prefix="d" uri="http://www.demo.org"/>
      </xsd:appinfo>
   </xsd:annotation>
   <xsd:element name="Demo">
      <xsd:annotation>
         <xsd:appinfo>
            <sch:pattern name="Check A greater than B">
               <sch:rule context="d:Demo">
                  <sch:assert test="d:A > d:B"
                           diagnostics="lessThan">
                        A should be greater than B.
                  </sch:assert>
               </sch:rule>
            </sch:pattern>
            <sch:diagnostics>
               <sch:diagnostic id="lessThan">
                  Error! A is less than B
                  A = <sch:value-of select="d:A"/>
                  B = <sch:value-of select="d:B"/>
               </sch:diagnostic>
            </sch:diagnostics>
         </xsd:appinfo>
      </xsd:annotation>
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="A" type="xsd:integer"/>
            <xsd:element name="B" type="xsd:integer"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Fonte: OGBUJI (2001).

Figura 3.1- Exemplo de esquema XSD na linguagem Schematron.

A Figura 3.1 mostra um exemplo de esquema XSD utilizando a linguagem Schematron. O objetivo desse documento XSD exemplo é restringir que o valor do elemento A seja maior que o valor do elemento B. Os elementos Schematron são identificados através do *namespace* sch, que é colocado dentro de um elemento annotation. Schematron extrai as informações de saída do documento XSD e cria um esquema Schematron, que é validado contra a instância do documento XML.

Outra abordagem, proposta por PROVOST (2002), para a imposição de restrições de domínio em documentos XML, se dá através da combinação de linguagens de estilo XSLT (XML *Stylesheet Language Transformations*) com *XPath*. PROVOST (2002) cita que, para impor restrições, é necessário uma linguagem para defini-las, como *XPath*, e um mecanismo para impô-las, como XSLT. Também afirma que essa abordagem é um estágio intermediário de validação. O primeiro seriam os esquemas (DTD e XSD) e o último seria através de escrita de código de validação com o uso de linguagens como C++ e Java. A Figura 3.2 mostra um exemplo de documento XSLT, que tem como objetivo validar duas restrições. A primeira verifica se existe um aparelho de som do tipo QuadraphonicDiscPlayer; caso exista, deve ter pelo menos quatro caixas de som disponíveis. A segunda verifica se existe um aparelho do tipo Stereo; caso exista, deve ter pelo menos uma caixa de som disponível.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:transform version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output method="text" />
    <xsl:strip-space elements="*" />
    <xsl:template match="text ()" />
        <xsl:template match="Stereo[QuadraphonicDiscPlayer][count (Speaker) < 4]" >
            <xsl:text>ERROR:  Quadraphonic sound source without enough speakers</xsl:text>
        </xsl:template>
        <xsl:template match="Stereo[count (CDPlayer | Turntable | CassetteDeck |
                QuadraphonicDiscPlayer) = 0]">
            <xsl:text>ERROR:  Stereo system must have at least one sound source</xsl:text>
    </xsl:template>
</xsl:transform>
```

Fonte: PROVOST (2002).

Figura 3.2- Exemplo de restrições usando XSLT e *XPath*.

BENEDIKT *et al.* (2002) descrevem uma outra linguagem de validação de restrições de integridade, chamada ΔX, e um compilador que produz um código de verificação dessas restrições. A linguagem é baseada em esquemas XSD, com expressões *XPath* e *XQuery*, e permite controlar uma maior variedade de restrições,

envolvendo: (i) valores de atributos, como por exemplo: o valor de um atributo A tem de ser maior que o valor de um atributo B; (ii) restrições referenciais, como por exemplo, um pedido deve apontar para um cliente com saldo maior que zero; (iii) associar cardinalidades com valores de atributos, como, por exemplo, o número de itens de um pedido não deve ser maior que o valor do atributo quantidade máxima de itens do pedido, estabelecida no elemento raiz.

A Figura 3.3 mostra um exemplo de definição de restrição na linguagem ΔX com auxílio de *XPath*. Essa restrição indica que para todo elemento region no esquema XSD, a capacidade máxima de uma região deve ser a mesma das regiões relacionadas.

A geração do código de verificação da linguagem ΔX é feita seguindo alguns passos. No primeiro, é traduzida a sintaxe do esquema para uma lógica interna de representação, que é compreendida pelo algoritmo de manipulação. No próximo passo, essa lógica é abstraída e é gerado um pseudocódigo procedural de classes. Ao final, o pseudocódigo é abstraído e traduzido para o código de implementação desejado.

```
<loop var="c" xpath=".">
    <loop var = "d" xpath = "c/AdjacentRegions/AdjacentRegion">
        <loop var = "e" xpath = "//Region[@nodeid=$d/@regionid]">
            <forbidden xpath="($e/@maxcapacity=$c/@maxcapacity)"/>
        </loop>
    </loop>
</loop>
```

Fonte: BENEDIKT *et al*. (2002).

Figura 3.3- Exemplo da linguagem ΔX.

BAYLEY *et al.* (2002) mostram uma abordagem de validação de restrições de integridade através de regras ECA (Evento-Condição-Ação), que realizam automaticamente ações em resposta a eventos. Uma linguagem é definida para especificar essas regras em repositórios XML, trabalhando somente com eventos para inserir e apagar elementos. Por exemplo, a Figura 3.4 (b) mostra uma regra ECA que atualiza o elemento livros_em_estoque no documento XML da Figura 3.4 (a) após um ou mais livros serem inseridos no documento XML.

Nos exemplos da Figura 3.4 (b) e Figura 3.4 (c), a variável do sistema $delta contém uma série de novos nodos inseridos detectados na parte "evento" da regra. A parte "condição" da regra verifica a cada novo livro se o valor do elemento

quantidade_estoque é maior que zero. Em caso positivo, uma cópia do elemento título é inserida como filho do elemento livros_em_estoque. A Figura 3.4 (c), similarmente, atualiza o elemento livros_em_estoque após a exclusão de um ou mais livros no documento.

```
<livraria>
    <livros_em_estoque>
        <título>The XML Schema Companion</título>
        <título>Perdas & Ganhos</título>
    </livros_em_estoque>
    <livro>
        <autor>
            <primeiro_nome>James</primeiro_nome>
            <nome_meio></nome_meio>
            <último_nome>Bayley</último_nome>
        </autor>
        <título>The XML Schema Companion</título>
        <data><ano>2004</ano></data>
        <isbn>0-456-200412-0</isbn>
        <quantidade_estoque>3</quantidade_estoque>
    </livro>
    <revista>
        <título>Exame</título>
        <data><dia>20</dia><mês>Junho</mês><ano>2004</ano></data>
    </revista>
    <livro>
        <autor>
            <primeiro_nome>Lya </primeiro_nome>
            <nome_meio></nome_meio>
            <último_nome> Luft </último_nome>
        </autor>
        <título>Perdas & Ganhos</título>
        <data><ano>2004</ano></data>
        <isbn>85-01-06711-3</isbn>
        <quantidade_estoque>2</quantidade_estoque>
    </livro>
</livraria>
```

(a)

```
on INSERT /livraria/livro
if $delta[quantidade_estoque>0]
do INSERT $delta/título BELOW /livraria/livros_em_estoque
```

(b)

```
on DELETE /livraria/livro
if TRUE
do DELETE título[.=$delta/título] BELOW /livraria/livros_em_estoque
```

(c)

Fonte: Adaptada de BAYLEY *et al.* (2002).

Figura 3.4- Exemplos de regras ECA.

Algumas conclusões importantes extraídas dos trabalhos mostrados devem ser consideradas.Todos os trabalhos que buscam incorporar restrições de domínio sugerem uma linguagem para tratar determinado problema. Através dela é gerado um código que efetua a validação das restrições com o documento (BENEDIKT *et al.*, 2002), ou são

incorporados comandos nos esquemas XML que são validados por um *parser* específico (OGBUJI, 2001).

Padrões de consulta *XPath* são, em geral, utilizados nas linguagens de validação e podem ser associados a uma linguagem de estilo XSLT para efetuar a validação de documentos XML (PROVOST, 2002).

BAYLEY *et al*. (2002) definem uma linguagem baseada em regras ECA para inserir e apagar elementos em documentos XML. Possui sintaxe semelhante a um gatilho (*trigger),* através da qual incorpora restrições de integridade baseadas em eventos em documentos XML, porém não é utilizada para tratar restrições de integridade de domínio.

Também existem trabalhos que tratam restrições integridade referencial e restrições de chaves, visando suprir algumas dificuldades encontradas em DTDs para tratamento de chaves, chaves estrangeiras, dependências funcionais e relacionamentos inversos (FAN & SIMEON, 1999). Também são tratadas questões referentes a chaves no mapeamento de documentos XML para bancos relacionais sobre a melhor maneira de efetuar a validação delas: se através do modelo relacional ou por meio de aplicações que verificam as restrições no documento XML. Além disso, BUNEMAN *et al*. (2002) trabalham com o conceito de chaves relativas[11] no contexto de bancos de dados científicos e estruturas hierárquicas. Porém, esses trabalhos não fazem parte do escopo desta dissertação.

O estudo de alguns bancos de dados nativos XML[12] também foi realizado, dentre os quais: Tamino XML Server, dbXML e Sonic XML Server 5.0 e outros (XMLDatabases, 2004). Analisando, de uma forma geral, esses bancos de dados XML nativos com relação à imposição de restrições de integridade, verifica-se que, além de integrarem os recursos já existentes da especificação XML, possuem interfaces que permitem a utilização de linguagens de programação, como, por exemplo, Java, para o tratamento adicional de restrições de integridade.

[11] Uma chave relativa consiste em um par (Q, K), onde Q é uma expressão de caminho e K é a chave.
[12] São bancos de dados que armazenam o documento XML de forma "nativa", geralmente como uma variante do mapeamento DOM para armazenamento de dados.

O banco dbXML, por exemplo, suporta gatilhos que são geralmente especificados através de classes Java que podem ser disparadas antes ou depois de uma inserção, atualização, remoção, ou recuperação de dados. Podem ser utilizadas para fazer a validação de documentos na inserção ou a modificação de documentos na recuperação. Portanto, pode-se afirmar que bancos de dados XML nativos não possuem recursos satisfatórios para tratar as questões levantadas na seção 3.2, uma vez que as restrições de integridade de domínio devem ser implementadas pela aplicação.

Além do estudo de bancos de dados nativos XML, foram analisados alguns projetos na área da Web Semântica, que surge como uma possível solução para a estruturação semântica dos dados na *web*.

A Web Semântica visa incorporar regras de semântica às informações, o que proporcionará que os usuários e as máquinas entendam as informações (BERNERS, 2001). Essas regras são especificadas através de ontologias[13], que permitem especificar explicitamente a semântica dos dados. Através dessas ontologias é possível elaborar uma rede enorme de conhecimento humano, complementando o processamento da máquina e melhorando quantitativamente o nível de servições na *web*.

Para que a Web Semântica se torne realidade são necessários alguns intrumentos, dentre os quais se pode citar o RDF (*Resource Description Framework*) e a OWL (*Ontology Web Language*). O RDF tem como objetivo definir um mecanismo de representação de metadados para descrever recursos não vinculados a um domínio específico de aplicação. A OWL apresenta mais recursos que o RDF e utiliza-se de uma lógica descritiva para explicitação do conhecimento, no entanto, ainda é pouco utilizada e existem poucas ferramentas e desenvolvedores capazes de lidar com esta linguagem.

Com base no estudo feito sobre a Web Semântica constatou-se que o uso de ontologias é indicado quando se deseja representar uma base de conhecimento, fornecendo informações sobre os dados disponíveis e especificando seus relacionamentos com o mundo real, não para imposição de restrições de integridade de domínio em documentos XML, que é o foco tratado por esta dissertação.

[13] Especificação explicita da interpretação estruturada de um domínio.

Assim, algumas análises são relevantes para o estabelecimento da proposta desta dissertação. A Tabela 3.5 compara os trabalhos relacionados com relação às restrições de integridade de domínio e restrições de integridade baseada em eventos do modelo relacional com o objetivo de verificar quais categorias do modelo relacional são tratadas pelos trabalhos relacionados

Tabela 3.5 - Comparativo trabalhos vs. relacional.

Trabalhos / **Relacional**	OGBUJI	PROVOST	BENEDIKT *et al.*	BAYLEY *et al.*
Domínio				
Tipo	√	√	√	-
Atributos	-	-	-	-
Tuplas	√	√	√	√
Banco de dados	√	√	√	√
Baseada em eventos	-	-	-	√

Fonte: Primária.

Analisando a Tabela 3.5, verifica-se que todos os trabalhos relacionados tratam as categorias de restrições de integridade do modelo relacional classificadas como tuplas e de banco de dados. Nenhum trabalho trata especificamente a categoria de restrições de integridade de atributo, pois esta já é tratada na especificação de esquemas XML. Somente um trabalho (BAYLEY *et al*.) possui um enfoque semelhante ao tratamento de restrições baseadas em eventos, porém este não possui um enfoque de tratamento de restrições de tipo.

A Tabela 3.6 compara os trabalhos relacionados com os recursos de imposição de restrições de integridade de domínio e restrições de integridade baseada em eventos da linguagem SQL, com o objetivo de apurar se algum trabalho relacionado utiliza-se de algum recurso semelhante ao existente na linguagem SQL. A justificativa para tal comparação é que a linguagem SQL oferece suporte completo, através de gatilhos (*TRIGGERS*), assertivas (*ASSERTIONS*) e da cláusula *CHECK*, para a especificação de restrições de integridade de domínio em bancos de dados relacionais, e pode ser

utilizada como base para a especificação de restrições de integridade de domínio em outros modelos de dados.

Na Tabela 3.6 a seguir, nota-se que 75% dos trabalhos relacionados possuem recursos semelhantes aos procedimentos existentes na linguagem SQL, ou seja, são instruções predefinidas e executadas através de uma chamada de aplicação. Apenas BAYLEY *et al*. trabalham com um recurso semelhante a gatilhos, no qual, em conseqüência da ocorrência de um evento e de uma determinada condição, são inseridos ou apagados elementos num documento XML. Nenhum dos trabalhos relacionados possui especificação semelhante a *checks* e transações existentes na linguagem SQL.As cláusulas *domain* e *assertion* podem ser tratadas pela especificação de esquemas XML e não fazem parte do escopo dos trabalhos.

Tabela 3.6- Comparativo trabalhos vs. SQL.

Trabalhos / **SQL**	OGBUJI	PROVOST	BENEDIKT *et al.*	BAYLEY *et al.*
Domain	-	-	-	-
Check	-	-	-	-
Assertion	-	-	-	-
Gatilhos	-	-	-	√
Procedimentos	√	√	√	-
Transações	-	-	-	-

Fonte: Primária.

Os trabalhos relacionados com relação a restrições de integridade de domínio do modelo relacional (Tabela 3.5) e com relação aos recursos de imposição de restrições de integridade de domínio da SQL (Tabela 3.6) demonstram algumas limitações, citadas a seguir:

- apenas um trabalho relacionado, de BAYLEY *et al*., apresenta um enfoque equivalente à maneira que a linguagem SQL trata restrições de integridade de domínio, porém serve apenas para inserir e apagar elementos em repositórios de dados. Nenhum dos outros trabalhos relacionados leva em conta este critério;

- algumas linguagens necessitam incorporar elementos predefinidos para controle de restrições de integridade nos esquemas XML. Para isso, é necessário o conhecimento aprofundado do esquema, além de que os *parsers* XML devem ser estendidos para validar as especificações das restrições de integridade impostas. Outra dificuldade refere-se ao entendimento do esquema, por conter elementos adicionais para a imposição de restrições;

- podem-se validar restrições de integridade de domínio em documentos XML pelo uso conjunto de XSLT e *XPath* (PROVOST), porém essa especificação tende a ser extensa, pois define um processamento complexo sobre o documento XML.

As limitações encontradas nos trabalhos relacionados motivaram o desenvolvimento desta dissertação, cuja proposta visa a um controle de restrições de integridade de domínio para documentos XML com base nos recursos da linguagem SQL para o tratamento dessas restrições para o modelo relacional, uma vez que a SQL atende adequadamente a essa categoria de restrição de integridade em bancos de dados relacionais. A proposta é apresentada no próximo capítulo, visando atender, além de outros, aos requisitos recém-mencionados.

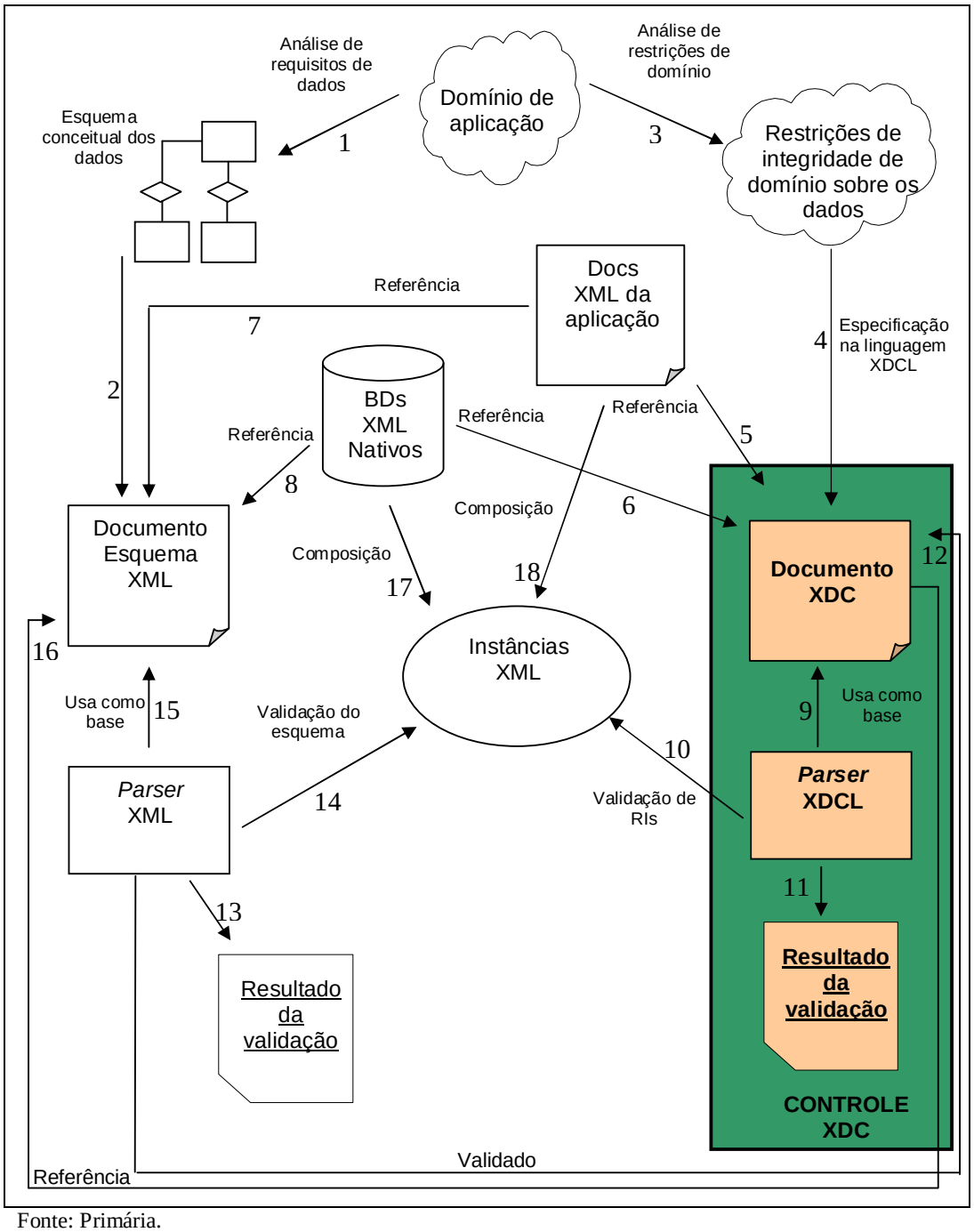
4 CONTROLE XDC (XML *Domain Constraint*)

Baseado na motivação descrita no capítulo 3, este capítulo apresenta um controle de restrições de integridade de domínio para documentos XML chamado XDC. Este controle é composto por uma linguagem chamada XDCL (XML *Domain Constraint Language*) e por um mecanismo de validação desta, chamado *parser* XDCL. A seguir são apresentadas a arquitetura de validação de documentos XML, sobre a qual o controle XDC se insere, a linguagem XDCL, o *parser* XDCL e o estudo de caso sobre o qual este controle foi aplicado.

4.1 Arquitetura

O principal objetivo desta dissertação é propor uma extensão para a tecnologia XML, visando ao tratamento de restrições de integridade de domínio em documentos XML, especificamente as restrições de integridade classificadas como restrições de tuplas, restrições de banco de dados e restrições de transição de estados que não são suportadas pela DTD e pelo XSD. Para permitir a validação de restrições de um documento XML, além das especificações do W3C, é necessário uma linguagem que defina restrições e uma ferramenta que valide essas restrições para um determinado documento XML.

O controle XDC proposto funciona como um recurso adicional de validação de restrições de domínio para aplicações que utilizam a tecnologia XML e pode ser adotado tanto numa aplicação que manipula documentos XML quanto num banco de dados XML nativo. A Figura 4.1 mostra a arquitetura de validação de documentos XML, levando em conta o controle XDC. Nele, a linguagem para a incorporação de restrições de integridade chama-se linguagem XDCL (XML *Domain Constraint Language*) e a ferramenta de validação dessas restrições chama-se *parser* XDCL.



Fonte: Primária.

Figura 4.1- Arquitetura XDC.

A Figura 4.1 apresenta um conjunto de setas numeradas que indicam a sua lógica de funcionamento. A seta 1 demonstra que, com base no domínio da aplicação, é feita a análise de requisitos dos dados e é gerado um esquema conceitual onde se especificam as restrições de integridade do domínio em questão. A seta de número 2 indica que,

através da utilização dos recursos existentes nos esquemas XML são impostas algumas restrições de integridade. A seta de número 3 indica que é realizado um levantamento das restrições de integridade de domínio que não foram consideradas pelos esquemas XML (DTD e XSD), tendo por conseqüência a seta de número 4, a qual demonstra que essas restrições são especificadas através do uso da linguagem XDCL e armazenadas num documento textual com extensão XDC.

Documentos XML e bancos de dados nativos XML que desejam estar de acordo com as restrições de integridade de um documento XDC fazem referência a ele, o que é indicado, respectivamente, pelas setas 5 e 6, assim como podem fazer referência a um documento esquema XML, mostrada nas setas 7 e 8.

A ferramenta que valida as restrições da linguagem XDCL contidas num documento XDC é o *parser* XDCL (seta 9), que valida as expressões XDCL contra as instâncias XML existentes (seta 10) num documento XML da aplicação ou de um banco de dados nativo XML (setas 18 e 17, respectivamente). Após ser feita a validação, o *parser* XDCL mostra o resultado da validação (seta 11), que pode ser uma mensagem de validação correta ou indicações de inconsistências existentes na instância XML.

Na arquitetura, a seta 12 indica que um *parser* XML valida o documento XDC, definido com base nas restrições de domínio da aplicação, verificando se é *válido* e *bem-formado*, e mostra o resultado dessa validação (seta 13). Também valida as instâncias XML (seta 14) com relação a um esquema XML (seta 15).

Um documento XDC referencia um esquema XSD (seta 16), visto que os elementos e atributos utilizados nas expressões da linguagem XDCL pertencem, necessariamente, a esse esquema, o qual funciona como um analisador sintático para o documento XDC. Também segue a sintaxe XML; portanto, nele podem ser aplicados recursos já existentes na tecnologia XML, como por exemplo, expressões de busca *XPath*.

Para ilustrar as restrições de integridade que a linguagem XDCL é capaz de impor, a Figura 4.2 mostra um documento XML cujo domínio é uma operadora de

planos de saúde que recebe dados de cobranças de serviços prestados de sua rede credenciada.

```
<dados>
  <instituicao>124</instituicao>
  <nome_instituicao>Hosp. Prontoclinicas Ltda</nome_instituicao>
  <prestadores>
    <prestador>1452</prestador>
    <nome_prestador>Hospital SP Ltda</nome_prestador>
    <responsavel>Maria da Silva</responsavel>
    <data_geracao mesgera="04">29/04/2004</data_geracao>
    <total_pagamentos>54.00</total_pagamentos>
      <consultas>
        <qtde_consultas>2</qtde_consultas>
        <total_consultas moeda="real">54.00</total_consultas>
        <consulta>
          <autorizacao>813321</autorizacao>
          <data_lancamento meslan="04">15/04/2004</data_lancamento>
          <paciente>14578</paciente>
          <nome_paciente>Adalgisa Severo</nome_paciente>
          <convenio>78</convenio>
          <nome_convenio>Saude Brasil Individual</nome_convenio>
          <medico>65</medico>
          <nome_medico>Adao Soares</nome_medico>
          <data_realizacao mesrea="04">12/04/2004</data_realizacao>
          <quantidade>1</quantidade>
          <valor_consulta moeda="real">27.00</valor_consulta>
        </consulta>
        <consulta>
          <autorizacao>81341</autorizacao>
          <data_lancamento meslan="04">12/04/2004</data_lancamento>
          <paciente>1245</paciente>
          <nome_paciente>Maria do Carmo</nome_paciente>
          <convenio>77</convenio>
          <nome_convenio>Saude Brasil Coletivo</nome_convenio>
          <medico>65</medico>
          <nome_medico>Adao Soares</nome_medico>
          <data_realizacao mesrea="04">20/04/2004</data_realizacao>
          <quantidade>1</quantidade>
          <valor_consulta moeda="real">27.00</valor_consulta>
        </consulta>
      </consultas>
  </prestadores>
</dados>
```

Fonte: Primária.

Figura 4.2- Exemplo de documento XML onde restrições de integridade podem ser definidas.

Um exemplo de restrição a ser considerada neste documento é que o conteúdo do elemento data_lancamento de uma consulta não pode ser uma data posterior ao conteúdo contido no elemento data_realizacao da mesma consulta. Este caso caracteriza uma restrição de integridade de tupla.

Outro tipo de restrição a ser considerada no mesmo documento é que o conteúdo do elemento total_pagamentos deve ser igual ao somatório dos conteúdos dos elementos valor_consulta. Este caso caracteriza uma restrição de integridade de banco de dados.

Para o tratamento dessas restrições, a linguagem XDCL utiliza como base recursos semelhantes aos existentes na linguagem SQL, principalmente através do uso de características semelhantes a gatilhos (*triggers*) e *checks*. A linguagem XDCL e o *parser* XDCL são vistos com mais detalhes a seguir.

4.2 Linguagem XDCL

Esta seção apresenta detalhadamente a linguagem XDCL (XML *Domain Constraint Language*), utilizada para incorporar restrições de integridade de domínio em documentos XML. Esse detalhamento é feito com base na BNF (*Backus Naur Form*) da linguagem XDCL utilizando exemplos sobre o documento mostrado na Figura 4.2.

Uma das principais características desta linguagem é o uso de uma sintaxe XML. Portanto, ela é definida através de um conjunto de elementos e atributos estruturados de forma hierárquica e armazenados num documento textual com a extensão XDC. Toda instância XML que deseja estar de acordo com as regras estabelecidas por um documento XDC deve ter um atributo com o nome xdcl no seu elemento raiz e cujo conteúdo contém a referência ao documento XDC com as restrições de integridade. Como exemplo, a Figura 4.3 mostra uma instância XML que faz referência a um documento XDC chamado restricoes.xdc.

```
<?xml version="1.0" encoding="UTF-8"?>
<dados xdcl="/projeto/restricoes.xdc">
        ...
</dados>
```

Fonte: Primária

Figura 4.3 – Exemplo de referência a um documento XDC.

Todos os elementos e atributos utilizados na linguagem XDCL devem, necessariamente, estar de acordo com um esquema XSD predefinido, mostrado com

detalhes no Anexo 4. Este esquema funciona como um analisador sintático para os elementos e atributos das cláusulas existentes na linguagem XDCL.

O principal objetivo da linguagem XDCL é executar uma ação, a qual pode ou não ser executada a partir de uma condição estabelecida. As ações podem ser o ato de inserir elementos e atributos, renomear elementos e atributos, apagar elementos e atributos e atualizar valores dos conteúdos de elementos e atributos ou mostrar uma mensagem informativa. Essas ações serão executadas por um mecanismo chamado *parser* XDCL instanciado, a partir de uma aplicação no momento estabelecido pelo usuário. A linguagem XDCL possui um conjunto de cláusulas, compostas por elementos e atributos que definem seus comandos. A Figura 4.4 mostra a BNF, que contém as cláusulas existentes na linguagem XDCL.

Conforme mostra a Figura 4.4, todo documento XDC inicia com o elemento <xdcl_constraints> e termina com o elemento </xdcl_constraints>, e o conteúdo entre esses elementos representa a(s) restrição(ões) de integridade que se deseja impor. Este elemento também contém a referência ao esquema XSD que valida os elementos e atributos da linguagem. Um exemplo desta referência é mostrado na Figura 4.5.

O conteúdo de um elemento xdcl_constraints pode conter um a vários elementos xdcl_constraint. Cada elemento xdcl_constraint representa uma restrição de integridade que se deseja impor. Este elemento possui um atributo xdcl_name cujo conteúdo indica o nome da restrição a qual ele representa. Por exemplo, a Figura 4.5 mostra um documento XDC que contém duas restrições de integridade, uma chamada testa_datas e a outra, testa_valores.

Um elemento xdcl_constraint possui dois subelementos: o primeiro é o elemento xdcl_on e o segundo, o elemento xdcl_statements. O elemento xdcl_on indica a qual elemento ou atributo do documento XML a restrição contida no elemento xdcl_constraint se aplica e seu conteúdo é uma expressão *XPath*, indicando o respectivo elemento ou atributo. Faz-se necessário o uso do elemento xdcl_on somente quando existirem condições a serem testadas.

```
<xdcl_constraints>
    {<xdcl_constraint xdcl_name="{nome_constraint}">
        [<xdcl_on>{Expressão Xpath}</xdcl_on>]
        {<xdcl_statements>
            [{<xdcl_set_conditions>
                {<xdcl_condition xdcl_operator="{= | >= | <= | > | < | <>}">
                    [<xdcl_old>
                        <xdcl_old_filexml>{caminho e nome do arquivo xml antigo}</xdcl_old_filexml>
                        <xdcl_old_identifier  type_id="{ElementDt|ElementNr|ElementSt}" [attr_identifier="{atributo}"]>{elemento}</xdcl_old_identifier>
                    </xdcl_old>]
                    {<xdcl_operand1 type_condition ="{ElementDt|ElementNr|ElementSt|ExpDt|ExpNr|ExpSt|ExpFuncNr}" [name_attr="{atributo}"] [old="{true}"]>
                        {expressão XPath | elemento}
                    </xdcl_operand1>
                    {<xdcl_operand2 type_condition ="{ElementDt|ElementNr|ElementSt|ExpDt|ExpNr|ExpSt|ExpFuncNr}" [name_attr="{atributo}"] [old="{true}"]>
                        {expressão XPath | elemento}
                    </xdcl_operand2>
                </xdcl_condition>
                [{<AND/> | <OR/> | <NOT/>}][...n]}[...n]
            </xdcl_set_conditions>}]
            {<xdcl_actions>
                {[<xdcl_delete>
                    <delete [name_attr= "{nome_atributo}"]>{nome delete}</delete>
                </xdcl_delete> |
                <xdcl_insert>
                    {<insert type= "{Element | Attribute}" name_element="{nome_elemento_pai}" [type_place="{Append | Before}"]>{nome insere}</insert>
                    [<insert-value>{valor insere}</insert-value>]}
                </xdcl_insert> |
                <xdcl_update>
                    <update name_element= "{nome elemento}" [name_attr="{nome atributo}" ] type_value="{Constant | Function}">
                        {função XPath | constante }
                    </update>
                </xdcl_update> |
                <xdcl_rename>
                    <rename name_element= "{nome_elemento}" [name_attr="{nome atributo}"]>{novo nome}</rename>
                </xdcl_rename> |
                <xdcl_message> {mensagem informativa }</xdcl_message>]}[1...n]
            </xdcl_actions>}
        </xdcl_statements>}
    </xdcl_constraint>}[...n]
</xdcl_constraints>
```

Fonte: Primária

Figura 4.4 – BNF da Linguagem XDCL

```
<xdcl_constraints xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="C:\xdcl\xdcl.xsd">
      <xdcl_constraint xdcl_name="testa_datas">
      ...
      </xdcl_constraint>
      <xdcl_constraint xdcl_name="testa_valores">
      ...
      </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.5 – Exemplo de um documento XDC

Caso exista o elemento xdcl_set_conditions, deve ter uma ou mais condições que são testadas e o seu resultado deve ser verdadeiro para que a ação possa ser executada pelo elemento xdcl_actions. Quando existir mais de uma condição, um dos elementos vazios AND, OR, NOT deve ser inserido após o elemento </xdcl_condition>, indicando a ocorrência dessa situação. Um exemplo onde são testadas duas restrições é mostrado na Figura 4.6. Nele, as datas só estarão consistentes se o valor do elemento data_lancamento for maior que o valor do elemento data_realizacao e o valor do atributo meslan do elemento data_lancamento for diferente do valor do atributo mesrea do elemento data_realizacao.

```
<xdcl_constraints>
   <xdcl_constraint xdcl_name=”testa_datas”>
      <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
      <xdcl_statements>
         <xdcl_set_conditions>
            <xdcl_condition xdcl_operator=”&gt;”>
               <xdcl_operand1 type_condition=”ElementDt”>data_lancamento</xdcl_operand1>
               <xdcl_operand2 type_condition=”ElementDt”>data_realizacao</xdcl_operand2>
            </xdcl_condition>
            <AND/>
            <xdcl_condition xdcl_operator=”&lt;&gt;”>
               <xdcl_operand1 type_condition=”ElementDt” name_attr=”meslan”>
                  data_lancamento
               </xdcl_operand1>
               <xdcl_operand2 type_condition=”ElementDt” name_attr=”mesrea”>
                  data_realizacao
               </xdcl_operand2>
            </xdcl_condition>
         </xdcl_set_conditions>
         <xdcl_actions>
            <xdcl_message>DATAS INCONSISTENTES</xdcl_message>
         </xdcl_actions>
      </xdcl_statements>
   </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.6 – Documento XDC com elemento AND.

Como citado anteriormente, um elemento xdcl_set_conditions pode ter uma ou várias condições; assim, cada condição é definida através do uso do elemento

xdcl_condition. Esse elemento contém os subelementos xdcl_operand1 e xdcl_operand2, além do atributo xdcl_operator. Também, dependendo da necessidade, pode haver o subelemento xdcl_old.

O atributo xdcl_operator do elemento xdcl_condition especifica qual operador lógico será utilizado como condição de comparação entre os valores dos elementos xdcl_operand1 e xdcl_operand2. A Tabela 4.1 mostra os possíveis valores para o atributo xdcl_operator, com seu valor equivalente na BNF da linguagem XDCL, seu significado e exemplo.

Tabela 4.1 – Valores do atributo xdcl_operator.

Condição	BNF	Significado	Exemplo
>	>	Maior	<xdcl_operand1> > <xdcl_operand2>
>=	>=	Maior ou igual	< xdcl_operand1> >= < xdcl_operand2>
<	<	Menor	< xdcl_operand1> < < xdcl_operand2>
<>	<>	Diferente	< xdcl_operand1> <> < xdcl_operand2>
<=	<=	Menor ou igual	< xdcl_operand1> <= < xdcl_operand2>
=	=	Igual	< xdcl_operand1> = < xdcl_operand2>

Fonte: Primária.

Os elementos xdcl_operand1 e xdcl_operand2 possuem os mesmos atributos e a mesma validação com relação ao seu conteúdo. O conteúdo desses elementos pode ser tanto o nome de um elemento quanto uma função *XPath* numérica que retorna o valor que se deseja testar.

O atributo type_condition associado a esses elementos é quem define o tipo do conteúdo do respectivo elemento. Os tipos de conteúdos podem ser um elemento ou um atributo do tipo de dados data, um elemento ou um atributo do tipo de dados numérico, um elemento ou um atributo do tipo de dados texto, um elemento ou um atributo relativo do tipo de dados data, um elemento ou atributo relativo do tipo de dados numérico, um elemento ou atributo relativo do tipo de dados texto ou uma função *XPath* numérica. A Tabela 4.2 mostra os possíveis valores do atributo type_condition com seu respectivo significado.

Algumas considerações são importantes quanto aos valores de conteúdo do atributo type_condition. Quando se define algum dos valores ElementDt, ElementNr, ElementSt para um elemento, este, obrigatoriamente, deve ser um elemento filho do

caminho especificado no elemento xdcl_on. Geralmente, utiliza-se esse tipo de definição para a imposição de restrições de integridade de domínio categorizadas como tupla, onde se testam dois elementos ou atributos pertencentes a uma mesma subárvore na hierarquia de elementos XML.

Tabela 4.2 – Valores do atributo type_condition nos elementos operand1 e operand2.

Valores	Significado
ElementDt	Elemento ou atributo do tipo de dados data.
ElementNr	Elemento ou atributo do tipo de dados numérico.
ElementSt	Elemento ou atributo do tipo de dados texto.
ExpDt	Elemento ou atributo relativo do tipo de dados data.
ExpNr	Elemento ou atributo relativo do tipo de dados numérico.
ExpSt	Elemento ou atributo relativo do tipo de dados texto.
ExpFuncNr	Função *XPath* numérica.

Fonte: Primária.

Como exemplo, deseja-se verificar se o elemento data_lancamento possui valor maior que o valor do elemento data_realizacao no contexto do elemento consulta. A Figura 4.7 mostra um exemplo de um documento XDC que impõe esta restrição usando a linguagem XDCL.

```
<xdcl_constraints>
    <xdcl_constraint xdcl_name=”testa_datas”>
        <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator=”&gt;”>
                    <xdcl_operand1 type_condition=”ElementDt”>data_lancamento</xdcl_operand1>
                    <xdcl_operand2 type_condition=”ElementDt”>data_realizacao</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>DATAS DE LANCAMENTO INCONSISTENTES</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.7 – Documento XDC com restrição de integridade de domínio - categoria tupla.

Caso o valor do atributo type_condition seja ExpDt, ExpNr ou ExpSt, o elemento especificado pode estar em qualquer nível da hierarquia a partir do caminho especificado no elemento xdcl_on. Esta situação é utilizada para a imposição de restrições de integridade de domínio categorizada como banco de dados, na qual se pode

testar o valor de um elemento ou atributo de uma determinada subárvore com o valor de um elemento ou atributo de uma outra subárvore.

Por exemplo, pode-se impor a restrição de que a data de geração do documento, elemento data_geracao, não pode ser menor que as datas de realização das consultas, elemento data_realizacao. A Figura 4.8 mostra um documento XDC que impõe essa restrição.

O elemento type_condition pode também ter o valor ExpFuncNr. Neste caso, o valor do conteúdo do respectivo elemento, xdcl_operand1 ou xdcl_operand2, deve ser uma função *XPath* numérica, a qual retornará um valor que será testado com a outra condição. Essa situação também caracteriza uma restrição de integridade de domínio de banco de dados.

```
<xdcl_constraints>
    <xdcl_constraint xdcl_name=”testa_data_geracao”>
        <xdcl_on>/dados/prestadores/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator=”&lt;”>
                    <xdcl_operand1 type_condition=”ExpDt”>data_geracao</xdcl_operand1>
                    <xdcl_operand2 type_condition=”ExpDt”>data_realizacao</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>DATA DE GERACAO INCONSISTENTE</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.8 – Documento XDC com restrição de integridade de domínio - categoria banco de dados.

Um exemplo de aplicação desse tipo de restrição é verificar se a soma dos valores das consultas (conteúdos dos elementos valor_consulta) é diferente do total dos pagamentos (conteúdo do elemento total_pagamentos). A Figura 4.9 ilustra este tipo de restrição de integridade.

Algumas considerações são importantes quanto aos valores de conteúdo do atributo type_condition. Quando se define algum dos valores ElementDt, ElementNr, ElementSt para um elemento, este, obrigatoriamente, deve ser um elemento filho do caminho especificado no elemento xdcl_on. Geralmente, utiliza-se esse tipo de

definição para a imposição de restrições de integridade de domínio categorizadas como tupla, onde se testam dois elementos ou atributos pertencentes a uma mesma subárvore na hierarquia de elementos XML.

```
<xdcl_constraints>
   <xdcl_constraint xdcl_name="testa_total_consultas">
      <xdcl_on>/dados/prestadores/*</xdcl_on>
      <xdcl_statements>
         <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
               <xdcl_operand1 type_condition="ElementNr">total_pagamentos</xdcl_operand1>
               <xdcl_operand2 type_condition="ExpFuncNr">
                  sum(/dados/prestadores/consultas/consulta/valor_consulta)
               </xdcl_operand2>
            </xdcl_condition>
         </xdcl_set_conditions>
         <xdcl_actions>
            <xdcl_message>TOTAL DE CONSULTAS INCONSISTENTE</xdcl_message>
         </xdcl_actions>
      </xdcl_statements>
   </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.9 – Documento XDC com restrição de integridade de domínio- categoria banco de dados utilizando uma função *XPath*.

Quando nos elementos xdcl_operand1 e xdcl_operand2 se necessita referenciar o conteúdo de um atributo para a imposição de restrições de integridade, deve-se anexar a esses elementos o atributo name_attr, cujo conteúdo corresponde ao nome do atributo do elemento que se deseja validar. Portanto, se existir o atributo name_attr, é feita a validação do atributo; em caso contrário, é feita a validação do elemento.

Para exemplificar esse tipo de situação, pode-se validar se o atributo moeda do elemento total_consultas possui o mesmo conteúdo do atributo moeda do elemento valor_consulta. A Figura 4.10 mostra um documento XDC que implementa essa restrição usando a linguagem XDCL.

Nos elementos xdcl_operand1 e xdcl_operand2 pode-se utilizar o atributo old. O uso deste atributo define uma função semelhantemente às cláusulas new e old utilizadas em gatilhos pela linguagem SQL e indica que o valor resultante do elemento é um valor antigo que pode ser comparado com o valor atual do mesmo elemento. Isso caracteriza as restrições de transição de estados.

```
<xdcl_constraints>
    <xdcl_constraint xdcl_name=”testa_moeda”>
        <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator=”=&lt;&gt;”>
                    <xdcl_operand1 type_condition=”ElementSt” name_attr=”moeda”>
                        total_pagamentos
                    </xdcl_operand1>
                    <xdcl_operand2 type_condition=”ExpSt” name_attr=”moeda”>
                        valor_consulta
                    </xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>ATRIBUTOS MOEDAS INCONSISTENTES</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.10 – Documento XDC com restrições de valores de atributos

Ao definir este tipo de restrição, além de utilizar o atributo old nos elementos xdcl_operand1 e xdcl_operand2, deve existir, como subelemento do elemento xdcl_condition, o elemento xdcl_old. Este elemento possui subelementos que indicam a localização da instância XML antiga e a identificação do elemento ou atributo correspondente da instância atual com a instância antiga. Isso é feito, respectivamente, através dos elementos xdcl_old_filexml e xdcl_old_identifier.

O conteúdo do elemento xdcl_old_filename indica a instância XML antiga, a qual contém os dados dos elementos e atributos que serão validados posteriormente com a instância atual. A manutenção de documentos XML antigos fica a cargo da aplicação responsável pela administração dos dados XML. Por exemplo, no contexto de um banco de dados XML nativo, toda vez que uma aplicação solicita um determinado documento XML é gerada uma cópia desse documento, que poderá ser validada posteriormente quando o documento XML atualizado retornar para ser armazenado no banco de dados nativo XML. A Figura 4.11 mostra a definição de um documento XDC que referencia um documento XML antigo através da especificação do caminho e do nome do documento XML referenciado (elemento xdcl_old_filename).

O elemento xdcl_old_identifier é responsável por identificar instâncias equivalentes de dados XML no documento XML antigo e novo para fins validação. Por exemplo, na Figura 4.11 o identificador é o elemento autorizacao e testa se o elemento

nome_paciente daquele código de autorização foi modificado com relação ao elemento nome_paciente do respectivo código de autorização da mesma instância no documento antigo.

```
<xdcl_constraints>
   <xdcl_constraint xdcl_name=”testa_codigo_prestador”>
      <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
      <xdcl_statements>
         <xdcl_set_conditions>
            <xdcl_condition xdcl_operator=”&lt;&gt;”>
               <xdcl_old>
                  <xdcl_old_filexml>/sistemas/bases/doc01.xml</xdcl_old_filexml>
                  <xdcl_old_identifier type_id=”ElementNr”>autorizacao</xdcl_old_identifier>
               </xdcl_old>
               <xdcl_operand1 type_condition=”ElementSt” old=”true”>
                  nome_paciente
               </xdcl_operand1>
               <xdcl_operand2 type_condition=”ElementSt”>
                  nome_paciente
               </xdcl_operand2>
            </xdcl_condition>
         </xdcl_set_conditions>
         <xdcl_actions>
            <xdcl_message>NOME DO PACIENTE FOI ALTERADO, VERIFIQUE</xdcl_message>
         </xdcl_actions>
      </xdcl_statements>
   </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.11 – Documento XDC com referencias a documentos XML antigos.

O elemento xdcl_old_identifier é responsável por identificar instâncias equivalentes de dados XML no documento XML antigo e novo para fins validação. Por exemplo, na Figura 4.11 o identificador é o elemento autorizacao e testa se o elemento nome_paciente daquele código de autorização foi modificado com relação ao elemento nome_paciente do respectivo código de autorização da mesma instância no documento antigo.

O atributo type_id deve ser anexado ao elemento xdcl_old_identifier com o propósito de definir o tipo de dado do elemento ou atributo identificador. Isso é necessário para que o *parser* XDCL possa efetuar as comparações corretamente. Os possíveis valores para o atributo type_id são ElementDt, ElementNr, ElementSt, cujos significados são descritos na Tabela 4.2.

No exemplo da Figura 4.11 o elemento identificador, autorização, é do tipo numérico. Caso haja a necessidade de referenciar um atributo como sendo o

identificador, pode-se adicionar ao elemento xdcl_old_identifier o atributo attr_identifier, cujo conteúdo é o nome do atributo que será o identificador.

A linguagem XDCL possibilita uma grande quantidade de combinações de condições entre os elementos xdcl_operand1 e xdcl_operand2. Alguns exemplos dessas combinações são mostrados no Anexo 5, os quais foram validados a partir do documento XML contido no Anexo 6; ambos os anexos fazem parte do estudo de caso.

O outro subelemento do elemento xdcl_statements é o elemento xdcl_actions, em cujo conteúdo estão especificadas as ações que são executadas contra a instância XML para garantia de integridade. Essas ações podem ser o ato de inserir elementos e atributos, apagar elementos e atributos, renomear elementos e atributos, atualizar os valores dos conteúdos de elementos e atributos ou mostrar uma mensagem informativa. Cada uma dessas ações possui um elemento representativo na linguagem XDCL. Esta associação é mostrada na Tabela 4.3.

Tabela 4.3 – Associação das ações e respectivos elementos na linguagem XDCL.

Ação	Elemento na linguagem XDCL
Apagar elementos e atributos.	xdcl_delete
Inserir elementos e atributos.	xdcl_insert
Atualizar elementos e atributos.	xdcl_update
Renomear elementos e atributos.	xdcl_rename
Mostrar mensagem informativa.	xdcl_message

Fonte: Primária.

De acordo com a BNF da linguagem XDCL (Figura 4.4), um elemento xdcl_actions pode ter um ou vários subelementos, os quais representam as ações que devem ser realizadas. Esses subelementos são vistos com mais detalhes a seguir.

O elemento xdcl_delete é utilizado quando se necessita remover elementos ou atributos. O conteúdo deste elemento especifica o nome do elemento que deve ser removido. O atributo name_attr pode ser anexado ao elemento xdcl_delete para indicar o nome do atributo, pertencente ao elemento especificado no conteúdo do elemento xdcl_delete, que deverá ser removido.

Como exemplo, a Figura 4.12 mostra um documento XDC que remove todos os atributos de nome moeda dos elementos que possuem o nome valor_consulta.

```
<xdcl_constraints>
    <xdcl_constraint xdcl_name=”apaga_atributo_moeda”>
        <xdcl_statements>
            <xdcl_actions>
                <xdcl_delete name_attr=”moeda”>valor_consulta</xdcl_delete>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.12 – Documento XDC com ação de remover atributos.

Outra ação possível de realizar através da linguagem XDCL é a inserção de elementos e atributos, indicada através do elemento xdcl_insert como subelemento do elemento xdcl_actions.

O elemento xdcl_insert deve ter o subelemento insert e pode ter o subelemento insert-value. O elemento insert é utilizado para definir o nome do elemento ou atributo que será inserido e o local onde será anexado.O conteúdo do elemento insert indica o nome do elemento ou atributo que será inserido.

O atributo type deve ser anexado ao elemento insert com o conteúdo “Element”, para indicar que um elemento será inserido, ou com o conteúdo “Attribute”, para indicar que um atributo será inserido. O atributo name_elemento também deve ser anexado ao elemento insert e o seu conteúdo indica o nome do elemento-pai do elemento ou atributo que será inserido, ou o nome do elemento referência, o qual indica que o novo elemento será inserido antes dele. No caso de inserção de elementos, pode-se utilizar o atributo type_place para indicar o local no documento XML onde o elemento será inserido. Se o conteúdo do atributo type_place for igual a Append, significa que o elemento será inserido após o último elemento filho do elemento-pai especificado no atributo name_element. Caso o conteúdo deste atributo seja igual a Before, o elemento será inserido antes do elemento especificado no atributo name_element. No caso de inserção de um atributo, este será inserido após o último atributo existente do elemento; nesta situação, não é necessário utilizar o atributo type_place.

Quando houver necessidade de informar o valor do conteúdo para o elemento ou atributo que se deseja inserir, pode-se utilizar o elemento insert-value, cujo valor indica esse conteúdo. A Figura 4.13 mostra um exemplo de um documento XDC que valida

uma condição e, caso esta seja verdadeira, efetua uma ação inserindo o atributo conferido com o valor do conteúdo igual a OK no elemento consulta.

```
<xdcl_constraints>
    <xdcl_constraint xdcl_name=”insere_atributos”>
        <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator=”=”>
                    <xdcl_operand1 type_condition=”ElementSt” name_attr=”moeda”>
                        total_pagamentos
                    </xdcl_operand1>
                    <xdcl_operand2 type_condition=”ExpSt” name_attr=”moeda”>
                        valor_consulta
                    </xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_insert>
                    <insert type=”Attribute” name_element=”consulta”>conferido</insert>
                    <insert-value>OK</insert-value>
                </xdcl_insert>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.13 – Documento XDC com ação de inserir atributos.

Pode-se usar o elemento xdcl_update como subelemento do elemento xdcl_actions, indicando que uma ação de atualização irá ocorrer. O elemento xdcl_update possui o subelemento update, cujo valor indica o novo conteúdo do elemento ou atributo a ser atualizado. O conteúdo deste elemento pode ser tanto um valor constante quanto uma expressão *XPath* numérica. Essa especificação de tipo de conteúdo é feita pelo conteúdo do atributo type_value, que deve ser anexado ao elemento update e pode conter os valores Function ou Constant para indicar, respectivamente, uma função ou uma constante.

A atualização é feita no elemento cujo nome é especificado no valor do atributo name_element anexado ao elemento update. Caso haja necessidade de atualizar valores de atributos, pode-se anexar ao elemento update o atributo name_attr, cujo conteúdo indica o nome do atributo que pertence ao elemento especificado no atributo name_element, o qual será atualizado com o valor do conteúdo do elemento update.

A Figura 4.14 mostra um exemplo de uma ação de atualização de um elemento. Nele é testada uma condição e, caso seja verdadeira, é atualizado o valor do elemento

total_consultas com uma função *XPath* numérica, que efetua a soma dos elementos valor_consulta.

```
<xdcl_constraints>
    <xdcl_constraint xdcl_name=”atualiza_valores”>
        <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator=”&lt;&gt;”>
                    <xdcl_operand1 type_condition=”ElementNr”>
                        total_pagamentos
                    </xdcl_operand1>
                    <xdcl_operand2 type_condition=”ExpFuncNr”>
                        sum(/dados/prestadores/consultas/consulta/valor_consulta)
                    </xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_update>
                    <update name_element=”total_consultas” type_value=”Function”>
                        sum(/dados/prestadores/consultas/consulta/valor_consulta)
                    </update>
                </xdcl_update>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.14 – Documento XDC com ação de atualização de elementos.

Outra ação que pode ser realizada pela linguagem XDCL é a modificação de um nome, de um elemento ou atributo para outro nome. Isso se dá através do uso do elemento xdcl_rename.

Esse elemento possui como subelemento o elemento rename, cujo conteúdo indica o novo nome do elemento ou atributo. O conteúdo do atributo name_element adicionado ao elemento rename indica o nome do elemento que será renomeado. Caso haja a necessidade de renomear um atributo, deve-se adicionar ao elemento rename o atributo name_attr, cujo conteúdo, em conjunto com o atributo name_element, indica o nome do atributo que deseja modificar.

Um exemplo de aplicação do elemento xdcl_rename é mostrado na Figura 4.15, que mostra um documento XDC que realiza a modificação do nome do elemento medico, passando para o nome codigo_medico.

Por fim, outra ação que pode ser executada através da linguagem XDC é o uso do subelemento xdcl_message, que mostra um texto informativo ao usuário da aplicação.

Um exemplo de aplicação desta ação é mostrado na Figura 4.15, onde, após executar a ação do elemento xdcl_rename, é mostrado um texto informativo ao usuário usando o elemento xdcl_message.

```
<xdcl_constraints>
   <xdcl_constraint xdcl_name=”renomeia_elementos”>
      <xdcl_statements>
         <xdcl_actions>
            <xdcl_rename>
               <rename name_element=”medico”>codigo_medico</rename>
            </xdcl_rename>
            <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO!</xdcl_message>
         </xdcl_actions>
      </xdcl_statements>
   </xdcl_constraint>
</xdcl_constraints>
```

Fonte: Primária

Figura 4.15 – Documento XDC com elementos xdcl_rename e xdcl_message.

4.3 Parser XDCL

O *parser* XDCL, como citado anteriormente, é o mecanismo utilizado para validar as condições e executar as ações XDCL contidas num documento XDC. Esta seção apresenta o algoritmo de validação da linguagem XDCL, que é executado pelo *parser* XDCL. Também nesta seção são feitas algumas considerações sobre a implementação do mesmo, embasadas na experiência adquirida durante o seu desenvolvimento.

O algoritmo XDCL é definido por meio de um conjunto de passos que indicam a lógica do funcionamento do *parser* XDCL, os quais aplicam recursivamente sobre o documento XML a ser validado e sobre o documento XDC que contém as restrições de integridade. Estes passos são vistos detalhadamente a seguir:

- Passo 1: O primeiro passo é efetuar a leitura da instância XML que contém os dados, buscando no elemento raiz do documento o atributo xdcl. O conteúdo deste atributo indica o caminho e o nome do arquivo XDC, o qual contém as restrições de integridade que deverão ser validadas;

- Passo 2: Caso o Passo 1 seja verdadeiro, deve-se instanciar o *parser* XML, que efetua a validação do documento XDC com relação às questões de *bem-formado* e

válido. Caso não ocorram erros, executa-se o Passo 3. Se existir algum outro aplicativo que faça essa validação, este passo pode ser descartado;

- Passo 3: Efetua-se a leitura do documento XDC que contém as restrições. Um arquivo XDC pode conter uma ou várias restrições. Cada uma deve ser tratada isoladamente; portanto, ao final da verificação de cada restrição, as variáveis utilizadas devem ser inicializadas para serem reutilizadas na próxima restrição;

- Passo 4: Busca-se o elemento xdcl_constraint, que indica a existência de uma restrição, e armazena-se o valor do atributo xdcl_name, o qual contém o nome da restrição que está sendo validada;

- Passo 5: Verifica-se a existência do elemento xdcl_on. Caso exista, armazena-se o valor do seu conteúdo, pois este indica em qual nível da instância XML ocorrerão as validações;

- Passo 6: Verifica-se a existência do elemento xdcl_set_conditions. Caso exista, executa-se o Passo 7 e, em caso contrário, executa-se o Passo 14;

- Passo 7: Verifica-se a existência dos elementos vazios AND, OR, NOT, que indicam a existência de mais de uma condição a ser testada e que o resultado deve estar conforme a cláusula especificada, para que a ação, ou as ações, seja executada;

- Passo 8: Efetua-se a leitura do elemento xdcl_condition e armazena-se o valor do atributo xdcl_operator, o qual especifica o tipo de comparação que será feita entre as condições;

- Passo 9: Verifica-se se existe o elemento xdcl_old, o qual indica referência a uma instância XML antiga. Caso exista, devem-se armazenar os valores dos elementos xdcl_old_file e xdcl_old_identifier e dos atributos type_id e attr_identifier. Em caso contrário, executa-se o Passo 10;

- Passo 10: Efetua-se a leitura do elemento xdcl_operand1 e armazenam-se o valor do seu conteúdo e o valor do conteúdo do atributo type_condition. Também se verifica

a existência do atributo name_attr, pois, se existir, indica uma referência a um atributo e também se deve armazenar o seu valor;

- Passo 11: Executam-se as mesmas ações do Passo 10 para o elemento xdcl_operand2;

- Passo 12: Efetua-se a leitura das instâncias XML correspondentes e validam-se as condições do Passo 7 ao Passo 11. Geralmente este passo é executado por um método da API SAX;

- Passo 13: Testam-se as condições do Passo 12;

- Passo 14: Executa-se a leitura do elemento xdcl_actions;

- Passo 15: Verifica-se que ações devem ser realizadas buscando os elementos xdcl_delete, xdcl_insert, xdcl_update, xdcl_rename e xdcl_message. Ao encontrar cada um, armazenam-se os valores de seus subelementos e atributos;

- Passo 16: Executa-se cada ação existente no Passo 15. Indica-se o uso de um método da API DOM para realizar as ações no documento.

Algumas considerações referentes ao algoritmo XDCL são importantes no momento da construção do *parser* XDCL. Analisando o algoritmo, verifica-se que, primeiramente, é feita uma leitura do documento XDC buscando as condições a serem testadas (Passo 6 ao Passo 12); depois, devem-se testar essas condições em relação à instância XML e verificar a sua validade. O mesmo acontece com as ações: primeiramente, é feita a leitura de cada ação do documento XDC (Passo 15) e, após, ela é executada (Passo 16).

Utilizou-se a API SAX para efetuar a leitura do documento XDC. Com isso, o método startElement foi utilizado para efetuar a verificação da existência de elementos e da existência e busca de valores de conteúdos de atributos. Para buscar o valor dos conteúdos dos elementos, utilizou-se o método characters. Para validar as condições ou executar as ações (Passo 12 e Passo 16), utilizou-se o método endElement.

Caso exista o elemento xdcl_on (Passo 5), é usada a API DOM, a qual permite que a instância XML seja filtrada a partir da expressão *XPath* contida no valor deste elemento, gerando, assim, em memória somente o fragmento da instância necessário para efetuar a validação. Esta mesma técnica pode ser utilizada com qualquer elemento ou atributo que utiliza expressões *XPath*.

O algoritmo XDCL pode ser implementado em qualquer linguagem de programação que suporte a tecnologia XML. Alguns exemplos de linguagens que podem ser utilizadas são Java, PHP, C++. Nesta dissertação foi desenvolvido um protótipo do *parser* XDCL, o qual foi implementado utilizando a linguagem Java. Tanto a linguagem XDCL como *parser* XDCL foram validados num estudo de caso, mostrado na próxima seção.

4.4 Estudo de caso

Esta dissertação foi validada no domínio de uma operadora de planos de saúde do Hospital Prontoclínicas Ltda, com sede na cidade de Passo Fundo-RS. Uma operadora de plano de saúde tem como objetivo prestar atendimento à saúde de seus usuários, como consultas, exames, internações e procedimentos médicos. Realizam esses atendimentos os prestadores de serviço, que, após o efetuarem, emitem uma cobrança para a operadora pelos serviços realizados.

Existe uma troca constante de dados dos prestadores de serviço (médicos, hospitais, laboratórios, clínicas, ...) com a operadora de planos de saúde. Essa troca de dados é feita através de documentos XML, que são validados por um esquema XSD e uma especificação XDC. Um exemplo de documento XML para este domínio é o mostrado no Anexo 6. Com essas validações minimiza-se o tempo desperdiçado com erros existentes nos documentos XML recebidos pela operadora de plano de saúde, garantindo uma maior consistência e correção dos dados recebidos.

A operadora possui um aplicativo responsável pelo gerenciamento da troca de dados, cujas funções são importar os documentos XML recebidos para o banco de dados relacional existente na operadora e também gerar instâncias XML desses documentos XML recebidos, para que possam ser exportadas e processadas por outras aplicações.

No caso de os documentos XML serem importados para o banco de dados relacional existente na operadora os documentos XML, já estão consistentes, minimizando a carga de validação semântica durante o processo de armazenamento. Em caso contrário, também é necessário que esses documentos XML estejam consistentes para que as instâncias geradas também o estejam.

Para efetuar a geração dos documentos XDC foi utilizada a ferramenta XMLSPY versão 2005 (ALTOVA, 2005), que possui recursos para a editoração e validação de documentos XML. Como o documento XDC é um documento XML, a utilização desta ferramenta tornou-se ideal para trabalhar com a geração desses documentos. Um dos principais recursos da ferramenta XMLSPY é a questão da validação do documento XDC com relação a ser *bem-formado* e *válido*.

O Anexo 5 mostra uma série de exemplos testados para o domínio da operadora de planos de saúde. Foi gerado um total de 52 exemplos com o objetivo de validar as possibilidades de restrições de integridade de domínio que podem ser impostas pela linguagem XDCL. Desses exemplos, 48 especificam condições que tratam as restrições de integridade de domínio definidas como tuplas, banco de dados e transição de estados, e 4 mostram as ações insert, rename, update, delete. Tais exemplos são baseados no documento mostrado no Anexo 6 e foram gerados no período de dezembro/2004.

O *parser* XDCL foi implementado a partir da linguagem Java, cuja escolha se justifica por funcionar em qualquer ambiente computacional, ser uma linguagem popular, ser ideal para desenvolvimento de aplicativos para a Internet e intranet e possuir uma grande gama de recursos disponíveis, dentre os quais o tratamento de exceções, tratamento de *strings* e outros (VELOSO, 2003). A versão utilizada na implementação do *parser* XDCL foi a j2skd1.4.2_06. O software JCreatorPro 2.5. (JCREATOR, 2004) foi o aplicativo de interface que, em conjunto com a linguagem Java, também foi utilizado no desenvolvimento.

Além dos métodos das APIs SAX e DOM, também foi utilizada na implementação do *parser* XDCL uma API de transformações TrAX (javax.xm.transform e seus subpacotes), cuja função é armazenar os documentos modificados num arquivo ou enviá-los para outra aplicação.

O uso do controle XDC (linguagem XDCL e *parser* XDCL) necessita ainda de um tempo maior de testes. Por enquanto, está sendo aplicado num projeto piloto utilizando um prestador de serviço. Porém, a expectativa por parte da equipe de desenvolvimento, em termos de afirmação e crescimento do seu uso, é otimista, considerando a necessidade de se manter dados XML consistentes, especificamente no âmbito de aplicações *web* que manipulam somente informações no formato XML.

5 CONCLUSÕES

Esta dissertação apresenta um controle de restrições de integridade de domínio para documentos XML chamado XDC, o qual é composto por uma linguagem para especificação de restrições de integridade e por um *parser* de validação, chamados, respectivamente, linguagem XDCL e *parser* XDCL.

A relevância desta dissertação está no fato de XML ser um padrão atual para representação e intercâmbio de dados na *Web* e a sua especificação de esquemas ser insuficiente para tratar todas as categorias de restrições de integridade existentes no modelo relacional, em particular restrições de integridade de domínio.A escolha do modelo relacional como base para estudo deve-se ao fato de este ser um modelo consolidado na comunidade de banco de dados e possuir um tratamento completo de restrições de integridade através da linguagem SQL.

Este capítulo apresenta as conclusões e alguns comparativos com respeito ao controle XDC. Todos esses pontos têm por objetivo salientar as suas contribuições e alguns trabalhos futuros mais significativos.

5.1 Contribuições

Algumas análises são importantes para demonstrar as contribuições desta dissertação. A primeira compara o controle XDC, mais a especificação de esquemas para documentos XML, com relação às categorias de restrições de integridade de domínio do modelo relacional, mostrada na Tabela 5.1.

Esta análise demonstra que, com o uso do controle XDC mais as especificações de esquemas para documentos XML, pode-se tratar todas as categorias de restrições de

integridade de domínio do modelo relacional, inclusive a categoria de transição de estados, que, neste caso, é considerada uma restrição de atributos.

Tabela 5.1 - Comparativo XDC e esquemas XML vs. relacional.

Controles / Relacional	XDC	Esquemas XML
Domínio		
Tipo	-	√
Atributos	√*	√
Tuplas	√	-
Banco de dados	√	-
Baseada em eventos	√	-

Fonte: Primária.

Além do tratamento das restrições de integridade de domínio, a Tabela 5.1 mostra também o tratamento de restrições de integridade baseadas em eventos, que é caracterizado pelo momento da chamada do *parser* XDCL pela aplicação, a qual ocorre num evento programado pelo usuário.

Outra análise importante é mostrada na Tabela 5.2, que exibe um comparativo do controle XDC e a especificação de esquemas XML com os recursos existentes na linguagem SQL para o tratamento de restrições de integridade de domínio.

Analisando a Tabela 5.2 verifica-se que a especificação de esquemas XML possui características semelhantes às clausulas *domain* e *assertion*, o que se dá através da definição de tipos de dados ou através do uso de *facets*. Verifica-se também que o controle XDC possui recursos semelhantes aos comandos *checks*, gatilhos e procedimentos. Isso se caracteriza, respectivamente, no controle XDC, pela possibilidade de especificar condições, executar ações e invocar o *parser* XDCL a partir de uma chamada feita pela aplicação na ocorrência de um evento. Nota-se também que, tanto no controle XDC quanto em esquemas XML, não existe controle de transações.

* Indica o tratamento de restrições de integridade de atributo classificadas como de transição de estados.

Esse assunto já está sendo pesquisado pela comunidade de banco de dados, porém não faz parte do escopo desta dissertação.

Tabela 5.2- Comparativo XDC e esquemas XML vs. SQL.

Controle / **SQL**	XDC	Esquemas XML
Domain	-	√
Check	√	-
Assertion	-	√
Gatilhos	√	-
Procedimentos	√	-
Transações	-	-

Fonte: Primária.

Por fim, a última análise feita é do controle XDC e trabalhos relacionados com relação aos recursos de imposição de integridade de domínio da linguagem SQL, mostrada na Tabela 5.3.

Tabela 5.3- Comparativo trabalhos relacionados e XDC vs. SQL.

Trabalhos / **SQL**	OGBUJI	PROVOST	BENEDIKT *et al*.	BAYLEY *et al.*	XDC
Domain	-	-	-	-	-
Check	-	-	-	-	√
Assertion	-	-	-	-	-
Gatilhos	-	-	-	√	√
Procedimentos	√	√	√	-	√
Transações	-	-	-	-	-

Fonte: Primária.

Analisando a Tabela 5.3, verifica-se que o controle XDC possui um maior número de recursos com características equivalentes às existentes na linguagem SQL, principalmente com características semelhantes a gatilhos e *checks*. Somente um trabalho relacionado possui características semelhantes a gatilhos e nenhum desses possui características semelhantes à cláusula *check*.

Com base nas análises anteriores e nas considerações feitas no decorrer da dissertação, pode-se afirmar que esta dissertação traz as seguintes contribuições:

- afirmação de que a especificação de esquemas XML é insuficiente para tratar todas as categorias de restrições de integridade do modelo relacional, principalmente a categoria de domínio;
- uma nova classificação de restrições de integridade para o modelo relacional, efetuada a partir da análise das obras de vários autores;
- um controle de restrições de integridade de domínio para documentos XML, principalmente com a imposição de restrições de integridade classificadas como tupla, banco de dados e transição de estados. Este controle chama-se XDC e utiliza características semelhantes a gatilhos e *checks* da SQL, com o objetivo de minimizar as deficiências do primeiro item;
- criação e implementação de um *parser* que efetua a validação das restrições de integridade, especificadas através do uso da linguagem XDCL e armazenadas num documento XDC.

5.2 Considerações finais

Esta dissertação é uma proposta de solução para o tratamento de restrições de integridade de domínio para documentos XML. Representa uma contribuição no sentido de preencher algumas lacunas existentes na especificação de esquemas XML com relação a restrições de integridade. Alguns trabalhos relacionados são encontrados na literatura visando atender a essas lacunas, embora demonstrem algumas limitações, como mostrado no capítulo 3.

Com base na afirmação dos trabalhos relacionados, esta dissertação propõe o controle XDC, que é composto pelo mecanismo de imposição, chamado linguagem XDCL, e pelo mecanismo de validação, chamado *parser* XDCL. O controle XDC possui como diferencial em relação aos trabalhos relacionados o fato de permitir um controle independente de integridade para documentos XML. A linguagem XDCL possui uma sintaxe XML; não necessita de elementos e atributos adicionais nos

esquemas XML, além de determinar um padrão XML para a imposição e validação das restrições, sem a necessidade de existirem códigos extensos e processamentos complexos sobre os documentos XML.

A linguagem XDCL possui funções semelhantes aos gatilhos e *checks* existentes na linguagem SQL, que possuem recursos (como especificações de condições e ações) para tratamento de restrições de integridade de tuplas, banco de dados e transição de estados, tratamento esse que é a proposta desta dissertação.

5.3 Trabalhos futuros

O controle XDC apresenta algumas limitações que podem ser tratadas e pode também sofrer algumas extensões. Essas atividades constituem trabalhos futuros, descritos a seguir:

- estender para trabalhar com transações, visto que este assunto não fez parte do escopo do controle XDC;
- otimizar através de testes em documentos maiores e menos estruturados, já que os testes realizados foram feitos em documentos XML, que contêm dados "fortemente" estruturados;
- implementar mecanismos de bloqueio e de controle de concorrência para as instâncias de documentos XML antigas, as quais necessitam ser validadas a partir de uma instância atual. Nesta dissertação, este tratamento está a cargo da aplicação que gerencia os dados XML;
- criar um mecanismo adicional para integração de expressões da linguagem *XUpdate* (XUpdate, 2004), que não foi considerada nesta dissertação devido a alguns problemas de validação ocorridos ao incorporar os elementos e atributos da *XUpdate* no esquema XSD da linguagem XDCL e pelo fato de *XUpdate* ainda estar se consolidando como uma linguagem para atualização de documentos XML;

- criar e implementar uma ferramenta para geração automática de documentos XDC, visto que nesta dissertação esses documentos foram gerados pela ferramenta XMLSPY 2005;

- implementar no *parser* XDCL a validação dos elementos vazios OR e NOT, os quais indicam, respectivamente, que uma ou outra condição deve ser verdadeira e se a negação da condição é verdadeira. Por enquanto, nesta versão do *parser* XDCL, foi implementada apenas a cláusula AND.

ANEXOS

ANEXO 1 -TIPOS SIMPLES X *FACETS*

Tipos simples	Facets					
	length	minLength	maxLength	pattern	enumeration	whiteSpace
string	y	y	y	y	y	y
normalizedString	y	y	y	y	y	y
token	y	y	y	y	y	y
byte				y	y	y
unsignedByte				y	y	y
base64Binary	y	y	y	y	y	y
hexBinary	y	y	y	y	y	y
integer				y	y	y
positiveInteger				y	y	y
negativeInteger				y	y	y
nonNegativeInteger				y	y	y
nonPositiveInteger				y	y	y
int				y	y	y
unsignedInt				y	y	y
long				y	y	y
unsignedLong				y	y	y
short				y	y	y
unsignedShort				y	y	y
decimal				y	y	y
float				y	y	y
double				y	y	y
boolean				y		y
time				y	y	y
dateTime				y	y	y
duration				y	y	y
date				y	y	y
gMonth				y	y	y
gYear				y	y	y
gYearMonth				y	y	y
gDay				y	y	y
gMonthDay				y	y	y
Name	y	y	y	y	y	y
QName	y	y	y	y	y	y
NCName	y	y	y	y	y	y
anyURI	y	y	y	y	y	y
language	y	y	y	y	y	y
ID	y	y	y	y	y	y
IDREF	y	y	y	y	y	y
IDREFS	y	y	y		y	y
ENTITY	y	y	y	y	y	y
ENTITIES	y	y	y		y	y
NOTATION	y	y	y	y	y	y
NMTOKEN	y	y	y	y	y	y
NMTOKENS	y	y	y		y	y

ANEXO 2 - TIPOS SIMPLES X *FACETS* ENUMERADAS.

Tipos Simples	**Facets**					
	max Inclusive	max Exclusive	min Inclusive	min Exclusive	totalDigits	fractionDigits
byte	y	y	Y	y	y	y
unsignedByte	y	y	Y	y	y	y
integer	y	y	Y	y	y	y
positiveInteger	y	y	Y	y	y	y
negativeInteger	y	y	Y	y	y	y
nonNegativeInteger	y	y	Y	y	y	y
nonPositiveInteger	y	y	Y	y	y	y
int	y	y	Y	y	y	y
unsignedInt	y	y	Y	y	y	y
long	y	y	Y	y	y	y
unsignedLong	y	y	Y	y	y	y
short	y	y	Y	y	y	y
unsignedShort	y	y	Y	y	y	y
decimal	y	y	Y	y	y	y
float	y	y	Y	y		
double	y	y	Y	y		
time	y	y	Y	y		
dateTime	y	y	Y	y		
duration	y	y	Y	y		
date	y	y	Y	y		
gMonth	y	y	Y	y		
gYear	y	y	Y	y		
gYearMonth	y	y	Y	y		
gDay	y	y	Y	y		
gMonthDay	y	y	Y	y		

ANEXO 3 – *FACETS* COM ELEMENTOS

Facets	Exemplos	Comentários
Length	<simpletype name=”strcpf”> <restriction base=”string”> <length value=”11”/> </restriction> </simpletypename>	Define que o número de caracteres dos elementos que referenciam o tipo “strcpf” é 11.
minLength, maxLength	<simpletype name=”strcpfcgc”> <restriction base=”string”> <minlength value=”11”/> <maxlength value=”14”/> </restriction> </simpletypename>	Define que o número mínimo de caracteres dos elementos que referenciam o tipo “strcpfcgc” é 11 e o máximo é 14.
Pattern	<simpletype name=”masvalor”> <restriction base=”string”> <pattern value=”\d\d\.\d{4}”/> </restriction> </simpletypename>	Define que os elementos que referenciam o tipo “masvalor” devem ter dois dígitos antes do ponto e quatro dígitos após o ponto. Ex: 10.0000
enumeration	<simpleType name=”tpnivel”> <restriction base=”NMTOKEN”> <enumeration value=”normal” /> <enumetation value=”secreto” /> <enumeration value=“muitosecreto” /> </restriction> </simpleType>	Define que os elementos que referenciam o tipo “tpnivel” podem possuir somente os seguintes valores: normal, secreto, muitosecreto.
minInclusive, maxInclusive	<simpletype name=”tpidade”> <restriction base=”integer”> <minInclusive value=”0”/> <maxInclusive value=”110”/> </restriction> </simpletypename>	Define que os valores válidos para os elementos que referenciam o tipo “tpidade” devem ser entre 0 e 110, inclusive eles.
minExclusive, maxExclusive	<simpletype name=”tpidade”> <restriction base=”integer”> <minExclusive value=”0”/> <maxExclusive value=”110”/> </restriction> </simpletypename>	Define que os valores válidos para os elementos que referenciam o tipo “tpidade” devem ser entre 0 e 110, exceto eles.
totalDigits, fractionDigits	<simpletype name=”tpdec”> <restriction base=”decimal”> <totalDigits value=”4”/> <fractionDigits value=”2”/> </restriction> </simpletypename>	Define que os valores válidos dos elementos devem ter um total de 4 dígitos sendo que 2 devem ser na parte fracionária.

ANEXO 4 – ESQUEMA XSD DA LINGUAGEM XDCL

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited  by Alexandre Tagliari Lazzaretti (Lazzaretti & Cia Ltda) -->
<xs:schema elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="xdcl_constraints">
<xs:complexType>
	<xs:sequence>
		<xs:element ref="xdcl_constraint" maxOccurs="unbounded"/>
	</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_constraint">
<xs:complexType>
	<xs:sequence>
		<xs:element ref="xdcl_on"/>
		<xs:element ref="xdcl_statements"/>
	</xs:sequence>
	<xs:attribute name="xdcl_name" type="xs:string" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_statements">
<xs:complexType>
	<xs:sequence>
		<xs:element ref="xdcl_set_conditions" minOccurs="0"/>
		<xs:element ref="xdcl_actions"/>
	</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_set_conditions">
<xs:complexType>
	<xs:choice maxOccurs="unbounded">
		<xs:element ref="xdcl_condition" maxOccurs="unbounded"/>
		<xs:element name="AND">
			<xs:complexType/>
		</xs:element>
		<xs:element name="OR">
			<xs:complexType/>
		</xs:element>
		<xs:element name="NOT">
			<xs:complexType/>
		</xs:element>
	</xs:choice>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_condition">
<xs:complexType>
	<xs:sequence>
		<xs:element ref="xdcl_old" minOccurs="0"/>
		<xs:element name="xdcl_operand1">
		<xs:complexType mixed="true">
		<xs:attribute ref="type_condition" use="required"/>
		<xs:attribute ref="name_attr" use="optional"/>
		<xs:attribute ref="old" use="optional"/>
		</xs:complexType>
		</xs:element>
		<xs:element name="xdcl_operand2">
		<xs:complexType mixed="true">
		<xs:attribute ref="type_condition" use="required"/>
		<xs:attribute ref="name_attr" use="optional"/>
		<xs:attribute ref="old" use="optional"/>
		</xs:complexType>
		</xs:element>
	</xs:sequence>
```

```
		<xs:attribute name="xdcl_operator" type="ope_log" use="required"/>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_old">
<xs:complexType>
		<xs:sequence>
			<xs:element name="xdcl_old_filexml" type="xs:string"/>
			<xs:element name="xdcl_old_identifier">
			<xs:complexType mixed="true">
			<xs:attribute name="type_id" type="tip_old" use="optional"/>
			<xs:attribute name="attr_identifier" type="xs:string" use="optional"/>
			</xs:complexType>
			</xs:element>
		</xs:sequence>
</xs:complexType>
</xs:element>
<xs:element name="xdcl_actions">
<xs:complexType>
		<xs:sequence>
			<xs:element ref="xdcl_delete" minOccurs="0"/>
			<xs:element ref="xdcl_insert" minOccurs="0"/>
			<xs:element ref="xdcl_rename" minOccurs="0"/>
			<xs:element ref="xdcl_update" minOccurs="0"/>
			<xs:element ref="xdcl_message" minOccurs="0"/>
		</xs:sequence>
</xs:complexType>
</xs:element>
<!-- COMEÇA A DEFINIÇÃO DAS CLÁUSULAS  DO ACTIONS-->
<!-- DELETE -->
<xs:element name="xdcl_delete">
<xs:complexType>
		<xs:sequence>
			<xs:element name="delete">
			<xs:complexType mixed="true">
			<xs:attribute ref="name_attr" use="optional"/>
			</xs:complexType>
			</xs:element>
		</xs:sequence>
</xs:complexType>
</xs:element>
<!-- INSERT -->
<xs:element name="xdcl_insert">
<xs:complexType>
		<xs:sequence>
			<xs:element name="insert">
			<xs:complexType mixed="true">
			<xs:attribute name="type" type="tip_ins" use="required"/>
			<xs:attribute name="name_element" type="xs:string" use="required"/>
			<xs:attribute name="type_place" type="tip_pla" use="optional"/>
			</xs:complexType>
			</xs:element>
			<xs:element name="insert-value" type="xs:string" minOccurs="0"/>
		</xs:sequence>
</xs:complexType>
</xs:element>
<!-- UPDATE -->
<xs:element name="xdcl_update">
<xs:complexType>
		<xs:sequence>
			<xs:element name="update">
			<xs:complexType mixed="true">
			<xs:attribute name="name_element" type="xs:string" use="required"/>
			<xs:attribute ref="name_attr" use="optional"/>
			<xs:attribute name="type_value" type="tip_up" use="required"/>
```

```
                    </xs:complexType>
                    </xs:element>
          </xs:sequence>
</xs:complexType>
</xs:element>
<!-- RENAME -->
<xs:element name="xdcl_rename">
<xs:complexType>
          <xs:sequence>
                    <xs:element name="rename">
                    <xs:complexType mixed="true">
                    <xs:attribute ref="name_element" use="required"/>
                    <xs:attribute ref="name_attr" use="optional"/>
                    </xs:complexType>
                    </xs:element>
          </xs:sequence>
</xs:complexType>
</xs:element>
<!-- MENSAGEM -->
<xs:element name="xdcl_message" type="xs:string"/>
<!-- DEFINIÇÃO DO ELEMENT -->
<xs:element name="xupdate_element">
<xs:complexType mixed="true">
          <xs:sequence>
                    <xs:element ref="xupdate_attribute" minOccurs="0"/>
          </xs:sequence>
          <xs:attribute ref="name" use="required"/>
</xs:complexType>
</xs:element>
<!-- DEFINIÇÃO DO ATTRIBUTE -->
<xs:element name="xupdate_attribute">
<xs:complexType mixed="true">
          <xs:attribute ref="name" use="required"/>
</xs:complexType>
</xs:element>
<!-- DEFINIÇÕES -->
<xs:attribute name="select" type="xs:string"/>
<xs:attribute name="name" type="xs:string"/>
<xs:attribute name="old" type="xs:string" fixed="TRUE"/>
<xs:attribute name="name_attr" type="xs:string"/>
<xs:element name="xdcl_on" type="xs:string"/>
<xs:attribute name="type_condition" type="tip_con"/>
<xs:attribute name="name_element" type="xs:string"/>
<!-- Definicao dos operadores lógicos -->
<xs:simpleType name="ope_log">
<xs:restriction base="xs:string">
          <xs:enumeration value="="/>
          <xs:enumeration value=">"/>
          <xs:enumeration value="&lt;"/>
          <xs:enumeration value=">="/>
          <xs:enumeration value="&lt;="/>
          <xs:enumeration value="&lt;>"/>
</xs:restriction>
</xs:simpleType>
<xs:simpleType name="tip_con">
<xs:restriction base="xs:string">
          <xs:enumeration value="ExpDt"/>
          <xs:enumeration value="ExpNr"/>
          <xs:enumeration value="ExpSt"/>
          <xs:enumeration value="ExpFuncNr"/>
          <xs:enumeration value="ElementDt"/>
          <xs:enumeration value="ElementNr"/>
          <xs:enumeration value="ElementSt"/>
</xs:restriction>
```

```
</xs:simpleType>
<xs:simpleType name="tip_old">
<xs:restriction base="xs:string">
        <xs:enumeration value="ElementDt"/>
        <xs:enumeration value="ElementNr"/>
        <xs:enumeration value="ElementSt"/>
</xs:restriction>
</xs:simpleType>
<!-- Definicao dos tipos dos inserts -->
<xs:simpleType name="tip_ins">
<xs:restriction base="xs:string">
        <xs:enumeration value="Element"/>
        <xs:enumeration value="Attribute"/>
</xs:restriction>
</xs:simpleType>
<!-- definição do conteúdo do atributo type_place -->
<xs:simpleType name="tip_pla">
<xs:restriction base="xs:string">
        <xs:enumeration value="Append"/>
        <xs:enumeration value="Before"/>
</xs:restriction>
</xs:simpleType>
<!-- Definicao dos tipos dos updates -->
<xs:simpleType name="tip_up">
<xs:restriction base="xs:string">
        <xs:enumeration value="Constant"/>
        <xs:enumeration value="Function"/>
</xs:restriction>
</xs:simpleType>
</xs:schema>
```

ANEXO 5 –EXEMPLO DE DOCUMENTO XDC

```
<?xml version="1.0" encoding="UTF-8"?>
<xdcl_constraints xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="C:\xdcl\xdcl.xsd">
      <! -- EXEMPLO 01 -->
    <xdcl_constraint xdcl_name="ATUALIZA_QUANTIDADES_CONSULTAS">
        <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
        <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade)</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr">qtde_consultas</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VALORES INCORRETOS VERIFIQUE</xdcl_message>
        </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
      <! -- EXEMPLO 02 -->
    <xdcl_constraint xdcl_name="ATUALIZACONSULTAS">
        <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
        <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementNr">qtde_consultas</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade)</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VALORES INCORRETOS, VERIFIQUE</xdcl_message>
        </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
      <! -- EXEMPLO 03 -->
    <xdcl_constraint xdcl_name="VERIFICADATAS">
        <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&gt;=">
                    <xdcl_operand1 type_condition="ElementDt">data_lancamento</xdcl_operand1>
                    <xdcl_operand2 type_condition="ElementDt">data_realizacao</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
```

```
            <xdcl_actions>
                <xdcl_message>DATAS INVALIDAS, VERIFIQUE !</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 04 -->
<xdcl_constraint xdcl_name="VERIFICA_MEDICOS">
        <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;&gt;">
                    <xdcl_operand1 type_condition="ElementSt">nome_medico</xdcl_operand1>
                    <xdcl_operand2 type_condition="ElementSt">medico_aut</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>NOMES DIFERENTES, VERIFIQUE!</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 05 -->
<xdcl_constraint xdcl_name="VERIFICAVALORES">
        <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;&gt;">
                    <xdcl_operand1 type_condition="ElementNr">valor_consulta</xdcl_operand1>
                    <xdcl_operand2 type_condition="ElementNr">valor_base</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>VERIFIQUE OS VALORES !</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 06 -->
<xdcl_constraint xdcl_name="VERIFICA_VALORES">
        <xdcl_on>/dados/prestadores/</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;&gt;">
                    <xdcl_operand1 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade)</xdcl_operand1>
                    <xdcl_operand2 type_condition="ExpNr">consultas/qtde_consultas</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
```

```
                <xdcl_message>VERIFIQUE OS VALORES !</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
     <! -- EXEMPLO 07 -->
    <xdcl_constraint xdcl_name="VERIFICA_QUANTIDADE_CONSULTAS">
        <xdcl_on>/dados/prestadores/</xdcl_on>
        <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpNr">consultas/qtde_consultas</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade)</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>VERIFIQUE OS VALORES !</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
     <! -- EXEMPLO 08 -->
    <xdcl_constraint xdcl_name="VERIFICA_VALORPAGO_VALORCONSULTA">
        <xdcl_on>/dados/prestadores/</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;&gt;">
                    <xdcl_operand1 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/valor_pago)</xdcl_operand1>
                    <xdcl_operand2 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/valor_consulta)</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>VERIFIQUE OS VALORES !</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
     <! -- EXEMPLO 09 -->
    </xdcl_constraint>
<!-- AS DATAS DE REALIZACAO TEM QUE SEREM MAIORES QUE A DATA BASE DAS CONSULTAS -->
    <xdcl_constraint xdcl_name="VERIFICADATAS">
        <xdcl_on>/dados/prestadores/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&gt;">
                    <xdcl_operand1 type_condition="ExpDt">data_base_consultas</xdcl_operand1>
                    <xdcl_operand2 type_condition="ExpDt">data_realizacao</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
```

```
            <xdcl_message>VERIFIQUE AS DATAS !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 10 -->
<xdcl_constraint xdcl_name="VERIFICA_VALORES_CONSULTAS">
    <xdcl_on>/dados/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpNr">valor_consulta</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpNr">valor_consulta_prestador</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DAS CONSULTAS !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 11 -->
<xdcl_constraint xdcl_name="VERIFICA_VALORES_CGCS">
    <xdcl_on>/dados/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementSt">cgc_prestador</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpSt">cgc_medico</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS CGCs !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 12 -->
<xdcl_constraint xdcl_name="VERIFICA_VALORES_CGCS">
    <xdcl_on>/dados/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementSt">cgc_prestador</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpSt">cgc_medico</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS CGCs !</xdcl_message>
```

```
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 13 -->
<xdcl_constraint xdcl_name="VERIFICADATAS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&gt;">
                <xdcl_operand1 type_condition="ElementDt">data_base_consultas</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpDt">data_realizacao</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE AS DATAS DE LANCAMENTO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 14 -->
<xdcl_constraint xdcl_name="VERIFICA_VALORES_CONSULTAS">
    <xdcl_on>/dados/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementNr">valor_consulta</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpNr">valor_consulta_prestador</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DAS CONSULTAS!</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 15 -->
<xdcl_constraint xdcl_name="VERIFICA_VALORES_CONSULTAS">
    <xdcl_on>/dados/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpNr">valor_consulta_prestador</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr">valor_consulta</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DE CONSULTAS!</xdcl_message>
        </xdcl_actions>
```

```
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 16 -->
<xdcl_constraint xdcl_name="VERIFICADATAS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&gt;">
                <xdcl_operand1 type_condition="ExpDt">data_base_consultas</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementDt">data_realizacao</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE AS DATA DE REALIZACAO DAS CONSULTAS!</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 17 -->
<xdcl_constraint xdcl_name="VERIFICA_VALORES_CONSULTAS">
    <xdcl_on>/dados/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpSt">cgc_prestador</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementSt">cgc_medico</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS CGCs !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 18 -->
<xdcl_constraint xdcl_name="VERIFICA_ATRIBUTOS_NUMEROS">
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementNr" name_attr="mes">data_lancamento</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr" name_attr="mes">data_realizacao</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS MESES DAS DATAS</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
```

```
</xdcl_constraint>
 <! -- EXEMPLO 19 -->
<xdcl_constraint xdcl_name="VERIFICA_ATRIBUTOS_STRINGS">
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementNr" name_attr="mes">data_lancamento</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr" name_attr="mes">data_realizacao</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS MESES DA DATA DE LANCAMENTO!</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 20 -->
<xdcl_constraint xdcl_name="VERIFICA_ATRIBUTOS_DATAS">
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&gt;">
                <xdcl_operand1 type_condition="ElementDt" name_attr="dtiniciorea">data_realizacao</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementDt" name_attr="dtiniciolan">data_lancamento</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE O ATRIBUTO DTINICIOREA !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 21 -->
<xdcl_constraint xdcl_name="VERIFICA_ATRIBUTOS_DATAS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpSt" name_attr="moeda_consulta">valor_consulta</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpSt" name_attr="moeda_prestador">valor_consulta_prestador</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DOS ATRIBUTOS MOEDA!</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
```

```
 <! -- EXEMPLO 22 -->
<xdcl_constraint xdcl_name="VERIFICA_ATRIBUTOS_DATAS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpNr" name_attr="mesgera">data_geracao</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpNr" name_attr="mes">data_lancamento</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DOS ATRIBUTOS</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 23 -->
<xdcl_constraint xdcl_name="VERIFICA_ATRIBUTOS_DATAS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&gt;">
                <xdcl_operand1 type_condition="ExpDt" name_attr="dt_base_inicio">data_base_consultas</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpDt" name_attr="dtiniciolan">data_lancamento</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DOS ATRIBUTOS DATAS</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 24 -->
<xdcl_constraint xdcl_name="ELEMENTOSATRIBUTOS_ELEMENTOSCAMINHOS_DATAS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;">
                <xdcl_operand1 type_condition="ElementDt" name_attr="dtiniciolan">data_lancamento</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpDt">data_base_consultas</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS ATRIBUTOS DAS DATAS DE LANCAMENTO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 25 -->
```

```
<xdcl_constraint xdcl_name="ELEMENTOSATRIBUTOS_ELEMENTOSCAMIHOS_NUMEROS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&gt;">
                <xdcl_operand1 type_condition="ElementNr" name_attr="qtde_min">qtde_consultas</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpNr">quantidade</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DAS QUANTIDADES!</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 26 -->
<xdcl_constraint xdcl_name="ELEMENTOSATRIBUTOS_ELEMENTOSCAMINHOS_STRINGS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementSt" name_attr="cpfcgc">paciente</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpSt">cgc_prestador</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DOS CGCs!</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 27 -->
<xdcl_constraint xdcl_name="ELEMENTOSCAMINHOS_ELEMENTOSATRIBUTOS_STRINGS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1      type_condition="ExpSt">cgc_prestador</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementSt" name_attr="cpfcgc">paciente</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DOS CGCs !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 28 -->
<xdcl_constraint xdcl_name="ELEMENTOSCAMINHOS_ELEMENTOSATRIBUTOS_DATAS">
```

```
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&gt;">
                <xdcl_operand1 type_condition="ExpDt">data_base_consultas</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementDt" name_attr="dtiniciolan">data_lancamento</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE O ATRIBUTO dtiniciolan !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 29 -->
<xdcl_constraint xdcl_name="ELEMENTOSCAMINHOS_ELEMENTOSATRIBUTOS_NUMEROS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;">
                <xdcl_operand1 type_condition="ExpNr">quantidade</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr" name_attr="qtde_min">qtde_consultas</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DAS QUANTIDADES DE CONSULTAS</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 30 -->
<xdcl_constraint xdcl_name="ELEMENTOSATRIBUTOS_CAMINHOSATRIBUTOS_NUMEROS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementNr" name_attr="mesgera">data_geracao</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpNr" name_attr="mes">data_lancamento</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE O ATRIBUTO mes DA DATA DE LANCAMENTO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 31 -->
<xdcl_constraint xdcl_name="ELEMENTOSATRIBUTOS_CAMINHOSATRIBUTOS_NUMEROS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
```

```
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementNr" name_attr="mesgera">data_geracao</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpNr" name_attr="mes">data_lancamento</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE O ATRIBUTO mes DA DATA DE LANCAMENTO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 32 -->
<xdcl_constraint xdcl_name="ELEMENTOATRIBUTO_CAMINHOATRIBUTO_DATAS">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ElementDt" name_attr="dt_base_inicio">data_base_consultas</xdcl_operand1>
                <xdcl_operand2 type_condition="ExpDt" name_attr="dtiniciolan">data_lancamento</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE O ATRIBUTO dtiniciolan DA DATA DE LANCAMENTO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 33 -->
<xdcl_constraint xdcl_name="CAMINHOATRIBUTO_ELEMENTOATRIBUTO_STRING">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpSt" name_attr="moeda_prestador">valor_consulta_prestador</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementSt" name_attr="moeda_consulta">valor_consulta</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DOS ATRIBUTOS MOEDA !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 34 -->
<xdcl_constraint xdcl_name="CAMINHOATRIBUTO_DATAS_ELEMENTOATRIBUTO">
    <xdcl_on>/dados/prestadores/*</xdcl_on>
    <xdcl_statements>
```

```
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;&gt;">
                    <xdcl_operand1 type_condition="ExpDt" name_attr="dtiniciolan">data_lancamento</xdcl_operand1>
                    <xdcl_operand2 type_condition="ElementDt" name_attr="dt_base_inicio">data_base_consultas</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>VERIFIQUE OS AS DATAS DE LANCAMENTO</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 35 -->
<xdcl_constraint xdcl_name="CAMINHOSATRIBUTOS_ELEMENTOSATRIBUTOS_NUMEROS">
        <xdcl_on>/dados/prestadores/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;=">
                    <xdcl_operand1 type_condition="ExpNr" name_attr="mes">data_lancamento</xdcl_operand1>
                    <xdcl_operand2 type_condition="ElementNr" name_attr="mesgera">data_geracao</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>VERIFIQUE OS VALORES DOS ATRIBUTOS mes !</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 36 -->
<xdcl_constraint xdcl_name="ELEMENTOATRIBUTO_FUNCAOATRIBUTO">
        <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;&gt;">
                    <xdcl_operand1 type_condition="ElementNr" name_attr="totqtde">qtde_consultas</xdcl_operand1>
                    <xdcl_operand2 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade)</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>VERIFIQUE OS VALORES DO ATRIBUTO totqtde!</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 37 -->
<xdcl_constraint xdcl_name="ELEMENTOATRIBUTO_FUNCAOATRIBUTO">
        <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
```

```
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade)</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr" name_attr="totqtde">qtde_consultas</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE O VALOR DO ATRIBUTO totqtde !!</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 38 -->
<xdcl_constraint xdcl_name="CAMINHOATRIBUTO_FUNCAO">
    <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="&lt;&gt;">
            <xdcl_operand1 type_condition="ExpNr" name_attr="totqtde">qtde_consultas</xdcl_operand1>
            <xdcl_operand2 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade/@qtde_minima)</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DOS ATRIBUTOS qtde_minima</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 39 -->
<xdcl_constraint xdcl_name="FUNCAO_CAMINHOATRIBUTO">
    <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="&lt;&gt;">
            <xdcl_operand1 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade/@qtde_minima)</xdcl_operand1>
            <xdcl_operand2 type_condition="ExpNr" name_attr="totqtde">qtde_consultas</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES DOS ATRIBUTOS qtde_minima !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 40 -->
<xdcl_constraint xdcl_name="FUNCAO_CAMINHOATRIBUTO">
    <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="&lt;&gt;">
```

```
            <xdcl_operand1 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade/@qtde_minima)</xdcl_operand1>
            <xdcl_operand2 type_condition="ExpNr" name_attr="totqtde">qtde_consultas</xdcl_operand2>
        </xdcl_condition>
        <AND/>
        <xdcl_condition xdcl_operator="&gt;">
            <xdcl_operand1 type_condition="ExpDt">data_base_consultas</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementDt" name_attr="dtiniciolan">data_lancamento</xdcl_operand2>
        </xdcl_condition>
        <AND/>
        <xdcl_condition xdcl_operator="&lt;&gt;">
            <xdcl_operand1 type_condition="ElementNr" name_attr="mesgera">data_geracao</xdcl_operand1>
            <xdcl_operand2 type_condition="ExpNr" name_attr="mes">data_lancamento</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VERIFIQUE OS VALORES !!!</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 41 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
    <xdcl_on>/dados/prestadores/consultas/</xdcl_on>
    <xdcl_statements>
        <xdcl_set_conditions >
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_old>
                    <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                    <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
                </xdcl_old>
                <xdcl_operand1 type_condition="ElementNr" old="TRUE"    >valor_pago</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr">valor_pago</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VALORES ALTERADOS PARA O VALOR PAGO, VERIFIQUE !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 42 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
    <xdcl_on>/dados/prestadores/consultas/</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
```

```
                    <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
                </xdcl_old>
                <xdcl_operand1 type_condition="ElementDt">data_realizacao</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementDt" old="TRUE">data_realizacao</xdcl_operand2>
            </xdcl_condition>
            <AND/>
            <xdcl_condition xdcl_operator="=">
                <xdcl_old>
                    <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                    <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
                </xdcl_old>
                <xdcl_operand1 type_condition="ElementSt">nome_convenio</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementSt" old="TRUE">nome_convenio</xdcl_operand2>
            </xdcl_condition>
            <AND/>
            <xdcl_condition xdcl_operator="=">
                <xdcl_old>
                    <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                    <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
                </xdcl_old>
                <xdcl_operand1 type_condition="ElementNr">valor_pago</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr" old="TRUE">valor_pago</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>ARQUIVO CONSISTENTE!</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 43 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
        <xdcl_on>/dados/prestadores/consultas/</xdcl_on>
        <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="=">
                <xdcl_old>
                    <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                    <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
                </xdcl_old>
                <xdcl_operand1 type_condition="ElementNr" name_attr="mes">data_realizacao</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr"  name_attr="mes" old="TRUE">data_realizacao</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>AS DATAS DE REALIZACAO ESTAO CONSISTENTES !</xdcl_message>
            </xdcl_actions>
```

```
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 44 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
        <AND/>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementSt" name_attr="cpfcgc" old="TRUE">paciente</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementSt"  name_attr="cpfcgc">paciente</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>O ARQUIVO ESTA CONSISTENTE</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 45 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
        <AND/>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
```

```
                    <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                    <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
                </xdcl_old>
                <xdcl_operand1 type_condition="ElementSt" old="TRUE">nome_paciente</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementSt">nome_paciente</xdcl_operand2>
            </xdcl_condition>
            <AND/>
            <xdcl_condition xdcl_operator="=">
                <xdcl_old>
                    <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                    <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
                </xdcl_old>
                <xdcl_operand1 type_condition="ElementDt" old="TRUE">data_lancamento</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementDt">data_lancamento</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_rename>
                    <rename name_element="data_lancamento">data_lan</rename>
                </xdcl_rename>
                <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO !</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
     <! -- EXEMPLO 46 -->
    <xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
        <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
        <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="=">
                <xdcl_old>
                    <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                    <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
                </xdcl_old>
                <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
            </xdcl_condition>
            <AND/>
            <xdcl_condition xdcl_operator="=">
                <xdcl_old>
                    <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                    <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
                </xdcl_old>
                <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
            </xdcl_condition>
```

```
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_delete>
                <delete>convenio</delete>
            </xdcl_delete>
            <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 47 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
        <AND/>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_insert>
                <insert type="Attribute" name_element="consulta">atr_cons</insert>
                <insert-value>22</insert-value>
            </xdcl_insert>
            <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 48 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
```

```
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
        <AND/>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_update>
                <update name_element="medico" type_value="Constant">200</update>
            </xdcl_update>
            <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 49 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
    <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="&lt;&gt;">
            <xdcl_operand1 type_condition="ElementNr" name_attr="totqtde">qtde_consultas</xdcl_operand1>
            <xdcl_operand2 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade)</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_update>
                <update name_element="qtde_consultas" name_attr="totqtde" type_value="Function">sum(/dados/prestadores/consultas/consulta/quantidade)</update>
            </xdcl_update>
            <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 50 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
```

```
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
        <AND/>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_delete>
                <delete name_attr="atr_cons">consulta</delete>
            </xdcl_delete>
            <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 51 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
        <AND/>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
```

```
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_rename>
                <rename name_element="autorizacao" name_attr="cod">cod_aut</rename>
            </xdcl_rename>
            <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
 <! -- EXEMPLO 52 -->
<xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
    <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
    <xdcl_statements>
    <xdcl_set_conditions>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
        <AND/>
        <xdcl_condition xdcl_operator="=">
            <xdcl_old>
                <xdcl_old_filexml>/xdcl/CasoOld.xml</xdcl_old_filexml>
                <xdcl_old_identifier type_id="ElementNr">autorizacao</xdcl_old_identifier>
            </xdcl_old>
            <xdcl_operand1 type_condition="ElementNr" name_attr="mes" old="TRUE">data_realizacao</xdcl_operand1>
            <xdcl_operand2 type_condition="ElementNr"  name_attr="mes">data_realizacao</xdcl_operand2>
        </xdcl_condition>
    </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_insert>
                <insert type="Element" name_element="prestadores" type_place="Append">codigo_prestador</insert>
                <insert-value>1</insert-value>
            </xdcl_insert>
            <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO !</xdcl_message>
        </xdcl_actions>
    </xdcl_statements>
</xdcl_constraint>
```

```
    <xdcl_constraint xdcl_name="ATUALIZA_QUANTIDADES_CONSULTAS">
        <xdcl_on>/dados/prestadores/consultas/*</xdcl_on>
        <xdcl_statements>
        <xdcl_set_conditions>
            <xdcl_condition xdcl_operator="&lt;&gt;">
                <xdcl_operand1 type_condition="ExpFuncNr">sum(/dados/prestadores/consultas/consulta/quantidade)</xdcl_operand1>
                <xdcl_operand2 type_condition="ElementNr">qtde_consultas</xdcl_operand2>
            </xdcl_condition>
        </xdcl_set_conditions>
        <xdcl_actions>
            <xdcl_message>VALORES INCORRETOS VERIFIQUE</xdcl_message>
        </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
    <xdcl_constraint xdcl_name="VERIFICA_MEDICOS">
        <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
        <xdcl_statements>
            <xdcl_set_conditions>
                <xdcl_condition xdcl_operator="&lt;&gt;">
                    <xdcl_operand1 type_condition="ElementSt">nome_medico</xdcl_operand1>
                    <xdcl_operand2 type_condition="ElementSt">medico_aut</xdcl_operand2>
                </xdcl_condition>
            </xdcl_set_conditions>
            <xdcl_actions>
                <xdcl_message>NOMES DIFERENTES, VERIFIQUE!</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
    <xdcl_constraint xdcl_name="ELEMENTONROLD_ELEMENTONR">
        <xdcl_on>/dados/prestadores/consultas/consulta/*</xdcl_on>
        <xdcl_statements>
            <xdcl_actions>
                <xdcl_insert>
                    <insert type="Element" name_element="medico" type_place="Before">teste</insert>
                    <insert-value>22</insert-value>
                </xdcl_insert>
                <xdcl_message>ATUALIZACAO REALIZADA COM SUCESSO !</xdcl_message>
            </xdcl_actions>
        </xdcl_statements>
    </xdcl_constraint>
</xdcl_constraints>
```

ANEXO 6 –EXEMPLO DE DOCUMENTO XML

```
<?xml version="1.0" encoding="UTF-8"?>
<dados xdcl="/xdcl/restricoes.xdc">
<instituicao>124</instituicao>
<nome_instituicao> Hosp. Prontoclinicas Ltda </nome_instituicao>
<prestadores>
<prestador>1452</prestador>
<nome_prestador>Hospital SP Ltda</nome_prestador>
<responsavel>Maria da Silva</responsavel>
<data_geracao mesgera="07">29/12/2004</data_geracao>
<total_pagamentos> 54.00 </total_pagamentos>
<valor_consulta_prestador moeda_prestador="dolar">25.00</valor_consulta_prestador>
<cgc_prestador>90619818000180</cgc_prestador>
<consultas>
<qtde_consultas totqtde="3" qtde_min="4">3</qtde_consultas>
<total_consultas>54.00</total_consultas>
<data_base_consultas dt_base_inicio="01/02/2005">01/01/2005</data_base_consultas>
<consulta>
        <autorizacao cod_aut="1">813321</autorizacao>
        <data_lancamento dtiniciolan="01/02/2004" mes="02">01/01/2004</data_lancamento>
        <paciente cpfcgc="90619818000180"> 14578 </paciente>
        <nome_paciente>Adalgisa Severo</nome_paciente>
        <convenio>78</convenio>
        <nome_convenio>Saude Brasil Individual</nome_convenio>
        <medico>200</medico>
        <nome_medico codigo_med="a01">Adao Soares</nome_medico>
        <cgc_medico>90619818000180</cgc_medico>
        <data_realizacao mes="03" dtiniciorea="02/02/2004">12/02/2004</data_realizacao>
        <quantidade qtde_minima="2">1</quantidade>
        <valor_consulta moeda_consulta="real">27.00</valor_consulta>
        <valor_base>27.00</valor_base>
        <medico_aut codigo_aut="a01">Adao Soares</medico_aut>
        <valor_pago>27.00</valor_pago>
</consulta>
<consulta>
        <autorizacao cod_aut="2">81341</autorizacao>
        <data_lancamento dtiniciolan="01/02/2004" mes="02" data_lan="">01/02/2004</data_lancamento>
        <paciente cpfcgc="9061981800018">1245</paciente>
        <nome_paciente>Maria do Carmo</nome_paciente>
        <convenio>12</convenio>
        <nome_convenio>Saude Brasil Coletivo</nome_convenio>
        <medico>200</medico>
        <nome_medico codigo_med="b02">Adao Soares</nome_medico>
        <cgc_medico>9061981800018</cgc_medico>
        <data_realizacao mes="05" dtiniciorea="02/05/2004">20/05/2004</data_realizacao>
        <quantidade qtde_minima="5">1</quantidade>
        <valor_consulta moeda_consulta="real">27.00</valor_consulta>
        <valor_base>30.00</valor_base>
        <medico_aut codigo_aut="b02">Adao Soares Lima</medico_aut>
        <valor_pago>26.00</valor_pago>
</consulta>
</consultas>
<codigo_prestador>1</codigo_prestador>
</prestadores>
</dados>
```

BIBLIOGRAFIA

1. ABITEBOUL, S.;BUNEMAN, P.;SUCIU, D. Data on the Web: From relations to Semistructured Data and XML. Ed. Morgan Kaufman, 2000.
2. ALTOVA, http://www.xmlspy.com/. Último acesso em janeiro de 2005.
3. ANDERSON, Richard; BIRBECK, Mark; KAY, Michael; LIVINGSTONE, Steven; LOESGEN, Brian; MARTIN, Didier; MOHR, Stephen; OZU, Nikola; PEAT, Bruce; PINNOCK, Jonathan; STARK, Peter; WILLIAMS, Kevin. Professional XML. Ed. Wrox Press Ltda, 2000. 1ª Edição.
4. ARCINIEGAS, Fabio; C++ XML. Ed. Makron Books, 2002. 1ª Edição
5. BAILEY, James; POULAVASSILIS, Alessandra; WOOD, Peter. Analisys and Optimization of Event-Condition-Action Rules on XML. *Computer Networks Journal - Special Issue on Web Dynamics*. Vol 39 (2002), 239-259
6. BENEDIKT, Michael; BRUNS, Glenn; GIBSON, Julie; KUSS, Robin; Amy Ng Bell Labs, Lucent Technologies. Automated Update Management for XML Integrity Constraints. Em Informal Proceedings of PLAN-X Workshop, 2002.
7. BENEDIKT, Michael; CHAN, Chee-Young; FAN, Wenfei; FREIRE, Juliana; RATOGI, Rajeev. Capturing types and constraints in Data Exchange. Em SIGMOD 2003.
8. BERNERS. T.B.; HENDLER; J., LASSILA, O. The Semantic Web. **Scientific American**, maio 2001. Disponível em: <http://www.scientificamerican.com/2001/050lissueberners-lee.html>
9. BRADLEY, Neil. The XML Companion. Ed.Addison Wesley, 2002. 3ª Edição.
10. BRADLEY, Neil. The XML Schema Companion. Ed.Addison Wesley, 2004. 1ª Edição.
11. BRADLEY, Neil. The XSL Companion. Ed.Addison Wesley, 2002. 2ª Edição.
12. BUNEMAN, Peter; DAVIDSON, Susan; FAN, Wenfei; HARA, Carmem; TAN, Wang-Chiew. Keys for XML. Disponível em: http://www.cse.ucsc.edu/~wctan/papers/2002/keysforxml-journal.pdf.

13. CARLSON, David. Modelagens De Aplicações XML com UML. Ed. Makron Books, 2002. 1ª Edição.
14. CASTRO, Elizabeth. XML para World Wide Web. Ed. Campus, 2001. 1ª Edição.
15. CHAUDHRI, Akmal; RACHID, Awais; ZICARI, Roberto. XML Data Management; Ed. Addison Wesley, 2003. 1ª Edição.
16. CHEN, Ye; DAVIDSON, Susan; ZHENG, Yifeng. Constraint Preserving XML Storage in Relations. Em VLDB, 2002. Disponível em www.db.ucsd.edu/webdb2002/papers/26.pdf
17. CHEN, Yi; DAVIDSON, Susan; ZHENG, Yifeng. Validating Constraints in XML. *Technical Report MS-CIS-02-03* (2002).
18. DATE, C. J. Introdução a Sistemas de Banco de Dados. Ed. Campus, 2000. 7ª Edição.
19. DATE, C.J. Bancos de Dados – Tópicos Avançados. Ed. Campus, 1998. 2ª Reimpressão.
20. DEUTSCH, Alin; TANNEN, Val. Containment and Integrity Constraints for Xpath Fragments. Em KRDB, 2001.
21. DONGWON, Lee; CHU, Wesley. Comparative Analysis of Six XML Schema Languages. Em ACM Sigmod Record, 29 (3), Setembro, 2000.
22. ELMASRI, R; NAVATHE, S. Fundamentals of Database Systens. Ed. Addison Wesley, 2000. 3ª Edição.
23. FAN, Wenfey; SIMÉON, Jérome. Integrity Constraints for XML. Em Symposium on Principles of Database Systems, 1999.
24. HEUSER, Carlos Alberto. Projeto de Banco de Dados. Ed. Sagra Luzzatto, 2001. 4ª Edição.
25. HOLZNER, Steven. Desvendando XML. Ed. Campus, 2001. 1ª Edição.
26. JCREATOR, http://www.jcreator.com. Último acesso em novembro de 2004.
27. MARCHAL, Benoît. XML by Example. Ed. Que, 2000. 1ª Edição.
28. NIEDERAUER, Juliano; PHP com XML – Guia de Consulta Rápida. Ed. Novatec, 2002.
29. OGBUJI, Chimizie. Validating XML with Schematron. Em http://www.xml.com/pub/a/2000/11/22/schematron.html

30. OLIVEIRA, Wilson José. Banco de Dados Interbase com Delphi. Ed. Visual Books, 2000. 1ª Edição.
31. PITTS-MOULTIS, Natanya; KIRK, Cheryl. XML Black Book. Ed. Makron Books, 2000. 1ª Edição.
32. PROVOST, Will. Beyond W3C XML Schema. Em http://www.xml.com/pub/a/2002/04/10/beyondwxs.html
33. SILBERCHATZ, Abraham; KORTH, Henry. F; SUDARSHAN, S. Sistemas de Banco de Dados. Ed. Makron Books, 1999. 3ª Edição.
34. SILVA, A. M.; Programando com XML. Ed. Campus, 2004. 1ª Edição.
35. TESCH, José Roberto. XML Shema. Ed. Visual Books, 2002. 1ª Edição.
36. VELOSO, Renê; JAVA E XML – Guia de Consulta Rápida. Ed. Novatec, 2003.
37. VILLARD, Lionel; LAYAÏDA, Nabil. An Incremental XSLT Transformation Processor for XML Document Manipulation. Em 12th Int'l World Wide Web Conf. (WWW'12), páginas 474-485, 2002.
38. W3C, eXtensible Markup Language. Disponível em: http://www.w3c.org. Último acesso em março de 2004.
39. W3Cnames, Disponível em: http://www.w3.org/TR/REC-xml-names/. Último acesso em fevereiro de 2004.
40. W3CXML, eXtensible Markup Language. Disponível em: http://www.w3c.org/xml. Último acesso em março de 2004.
41. W3CXML1.1, http://www.w3.org/TR/2004/REC-xml11-20040204/. Último acesso em março de 2004.
42. W3CXML2003, http://www.w3.org/2001/03/XMLSchema/ Último acesso em julho de 2004.
43. W3CXMLLAN, http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/#language. Último acesso março 2004.
44. W3CXMLSchema Disponível em: http://www.w3.org/XML/Schema#dev. Último acesso em fevereiro 2004.
45. W3CXMLTR-0, http://www.w3.org/TR/xmlschema-0/ Último acesso em março de 2004.
46. W3CXMLXPath, http://www.w3.org/TR/XPath. Último acesso em Julho de 2004.

47. XMLDatabases, http://www.rpbourret.com/xml/XMLDatabaseProds.htm. Último acesso em dezembro de 2004.
48. XUpdate, http://xmldb-org.sourceforge.net/xupdate/xupdate-wd.html. Último acesso em janeiro de 2005.