CONTROLE DE RESTRIÇÕES DE INTEGRIDADE DE DOMÍNIO EM DOCUMENTOS XML

Trabalho Individual submetido 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, Agosto de 2004
<table>
<thead>
<tr>
<th>Abreviação</th>
<th>Significado</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>DOM</td>
<td>Document Object Model</td>
</tr>
<tr>
<td>DTD</td>
<td>Document Type Definition</td>
</tr>
<tr>
<td>EBNF</td>
<td>Extended Backus Naur Form</td>
</tr>
<tr>
<td>ECA</td>
<td>Event Condition Action</td>
</tr>
<tr>
<td>SAX</td>
<td>Simple API for XML</td>
</tr>
<tr>
<td>SQL</td>
<td>Structred Query Language</td>
</tr>
<tr>
<td>URI</td>
<td>Uniform Resource Identifier</td>
</tr>
<tr>
<td>W3C</td>
<td>World Wide Web Consortium</td>
</tr>
<tr>
<td>XDC</td>
<td>XML Domain Constraint</td>
</tr>
<tr>
<td>XDCL</td>
<td>XML Domain Constraint Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
<tr>
<td>XSD</td>
<td>XML Schema Definition</td>
</tr>
<tr>
<td>XSL</td>
<td>XML Stylesheet Language</td>
</tr>
<tr>
<td>XSLT</td>
<td>XML Stylesheet Language Transform</td>
</tr>
</tbody>
</table>
Lista de Figuras

Figura 2.1- Exemplo de restrição de domínio................................................................. 16
Figura 2.2- Exemplo de restrições básicas de tabelas.................................................. 16
Figura 2.3- Exemplo de restrição geral........................................................................ 17
Figura 2.4- Exemplo de gatilho..................................................................................... 17
Figura 2.5- Exemplo de procedimento......................................................................... 18
Figura 2.6- Exemplo de transação................................................................................ 19
Figura 3.1- Exemplo de um documento XML ............................................................ 23
Figura 3.2- Exemplo de DTD........................................................................................ 26
Figura 3.3- Exemplo de XSD........................................................................................ 32
Figura 3.4- Exemplo de elemento com conteúdo vazio............................................... 33
Figura 3.5- Exemplo de elemento com enumeração de valores possíveis.................... 34
Figura 3.6- Exemplo de elemento all............................................................................ 36
Figura 3.7- Exemplo do elemento any.......................................................................... 36
Figura 3.8- Exemplo do elemento group...................................................................... 37
Figura 3.9- Exemplos de declarações de atributos....................................................... 38
Figura 3.10- Exemplo de definição de listas................................................................. 41
Figura 3.11- Exemplo de referências de chaves............................................................ 45
Figura 4.1- Classificação de linguagens de esquemas XML...................................... 55
Figura 4.2- Exemplo de esquema XML na linguagem Schematron.............................. 56
Figura 4.3- Exemplo de restrições usando XSLT e XPath......................................... 57
Figura 4.4- Exemplo da linguagem ΔX.......................................................................... 58
Figura 4.5- Exemplos de regras ECA........................................................................... 59
Figura 5.1- Arquitetura da abordagem proposta......................................................... 64
Figura 5.2- Exemplo de documento XML onde restrições de integridade podem ser definidas. 66
Lista de Tabelas

Tabela 2.1 - Comparativo relacional vs. SQL................................................................. 20
Tabela 3.1- Tipos de declarações de marcação............................................................. 25
Tabela 3.2- Tipos de atributos..................................................................................... 29
Tabela 3.3 - Padrões de atributos............................................................................... 30
Tabela 3.4- Entidades predefinidas............................................................................. 30
Tabela 3.5 - Tipos de facets....................................................................................... 40
Tabela 3.6- Tipos de facets enumeradas..................................................................... 40
Tabela 3.7- facets com elementos.............................................................................. 41
Tabela 3.8- Expressões XPath..................................................................................... 46
Tabela 4.1 - Comparativo DTD vs relacional............................................................. 49
Tabela 4.2- Comparativo XSD vs relacional (I).......................................................... 51
Tabela 4.2- Comparativo XSD vs relacional (II)......................................................... 52
Tabela 4.3- Comparativo trabalhos vs. relacional..................................................... 60
Tabela 4.4- Comparativo trabalhos vs. SQL............................................................... 61
Tabela 5.1.- Cronograma de atividades....................................................................... 68
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. Estas características o tornam adequado a 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, são feitos um levantamento das restrições de integridade existentes no modelo de dados relacional 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 sendo, esta dissertação propõe um mecanismo de controle de restrições de integridade de domínio, inexistentes em esquemas DTD e XSD, para documentos XML e bancos de dados nativos XML. Este mecanismo 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. A linguagem XDCL utiliza como base a maneira que a linguagem SQL trabalha com restrições de integridade no modelo relacional, através de comandos semelhantes a triggers, assertions e checks.

Palavras-chave: restrições de integridade, SQL, XML, DTD, XSD.
Sumário

Lista de Abreviaturas e Siglas ................................................................. 2
Lista de Figuras ..................................................................................... 3
Lista de Tabelas .................................................................................... 4
Resumo .................................................................................................. 5
1 Introdução ......................................................................................... 7
  1.1 Apresentação ............................................................................... 7
  1.2 Objetivos ..................................................................................... 7
  1.3 Justificativas ............................................................................... 9
  1.4 Organização dos capítulos .......................................................... 9
2 Restrições de Integridade em Bancos de Dados Relacionais e SQL .......... 11
  2.1 Conceitos .................................................................................. 11
  2.2 Restrições de integridade em bancos de dados relacionais ............... 12
    2.2.1 Restrições de domínio ......................................................... 12
    2.2.2 Restrições de chaves .......................................................... 14
    2.2.3 Restrições de integridade referencial .................................. 14
    2.2.4 Restrições quanto ao momento da verificação ....................... 15
    2.2.5 Restrições baseadas em eventos ......................................... 15
  2.3 Restrições de integridade em SQL ............................................... 16
  2.4 Consideração de restrições de integridade relacionais por SQL ........ 20
  2.5 Considerações finais .................................................................. 21
3 XML .................................................................................................. 22
  3.1 Tecnologia XML ........................................................................ 22
  3.2 Dados XML e Documentos XML ............................................... 22
  3.3 Esquemas XML ......................................................................... 24
    3.3.1 DTD ................................................................................ 25
    3.3.2 XSD .............................................................................. 31
  3.4 Considerações finais .................................................................. 47
4 Restrições de Integridade para Dados XML ...................................... 48
  4.1 Relacional vs. Esquemas XML ..................................................... 48
    4.1.1 Relacional vs. DTD ........................................................... 49
    4.1.2 Relacional vs. XSD .......................................................... 51
  4.2 Trabalhos Relacionados ............................................................. 54
5 Incorporação de Restrições de Integridade de Domínio em Documentos XML .... 63
  5.1 Arquitetura da Abordagem .......................................................... 63
  5.2 Estudo de caso .......................................................................... 67
  5.3 Atividades futuras ...................................................................... 67
Referências .......................................................................................... 69
Anexos ............................................................................................... 72
1 Introdução

O termo “integridade” refere-se à precisão ou correção de dados em um banco de dados (DATE, 2000). Um dos objetivos primordiais de um sistema gerenciador de banco de dados é a integridade dos dados. Dizer que os dados em um banco de dados estão íntegros significa dizer que eles refletem a realidade representada pelo banco de dados e que são consistentes entre si (HEUSER, 2001). Para garantir a integridade de um banco de dados, são necessárias regras que estabeleçam a consistência dos dados no banco de dados.

XML (eXtensible Markup Language) é um padrão atual para representação e intercâmbio de dados na Web (W3CXML, 2004). Com ele é possível criar estruturas de dados arbitrárias e, então, compartilhá-las com outros aplicativos.

1.1 Apresentação

Devido ao crescente uso da tecnologia XML para a troca de dados, a pesquisa e o desenvolvimento de bancos de dados para XML e o importante papel que as restrições de integridade desempenham em busca da consistência, precisão ou correção dos dados, tornam-se necessários mecanismos que imponham restrições de integridade em documentos XML.

Mais especificamente, busca-se tratar a questão de restrições de integridade de domínio, visto que, com base no estudo, notou-se que é preciso impor restrições semânticas além das que os esquemas XML (DTD e XSD) oferecem.

1.2 Objetivos

Portanto, esta dissertação procura atender aos seguintes objetivos:
**Objetivo Geral**

Propor 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.

**Objetivos Específicos**

Os objetivos específicos desse trabalho são:

- analisar as restrições de integridade que a teoria do modelo relacional estabelece e definir uma classificação para elas;
- verificar a maneira como a linguagem SQL trata essas restrições de integridade do modelo relacional estabelecidas na classificação anterior;
- analisar a tecnologia XML, dando ênfase às recomendações para definição de esquemas (DTD e XSD), mostrando, com base na classificação estabelecida para o modelo relacional, quais são as restrições de integridade que estas recomendações impõem;
- levantar as limitações dos esquemas XML para a definição de restrições de integridade de domínio;
- verificar trabalhos relacionados que buscam suprir essas limitações;
- propor uma especificação de restrições de integridade de domínio para esquemas XML que ainda não foram consideradas ou foram consideradas parcialmente por propostas de trabalhos relacionados. Para isso, utiliza-se como base a maneira que a linguagem SQL as trata, levando em conta seu poder de expressão para definição de restrições de integridade em bancos de dados relacionais;
- propor uma ferramenta de validação de restrições de integridade de domínio para dados XML que estão de acordo com um esquema XML para o qual as restrições se aplicam.
1.3 Justificativas

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.


Esta dissertação propõe uma arquitetura para controle de restrições de integridade de domínio em documentos XML. Esta arquitetura é composta por uma linguagem de definição de restrições chamada XDCL (XML Domain Constraint Language) e por um parser de validação de restrições chamado parser XDCL. Seus pontos fortes são: (i) independência do contexto XML; (ii) referência a esquemas XML, portanto utilizando seus mecanismos de imposição de restrições de integridade; (iii) sintaxe XML; (iv) tratamento de restrições de integridade de tuplas e banco de dados, inexistentes em esquemas XML.

1.4 Organização dos capítulos

O presente trabalho apresenta outros quatro capítulos. O capítulo 2 apresenta uma classificação de restrições de integridade no modelo relacional e um estudo de como a linguagem SQL impõe restrições de integridade. Ao final, é feito um comparativo mostrando se a linguagem SQL atende todas as restrições de integridade existentes no modelo relacional e de que maneira ela as impõe.
No capítulo 3 é feito o estudo da tecnologia XML. Nele são apresentados três padrões da tecnologia: *namespaces*, DTD e XSD. O objetivo do capítulo é mostrar os recursos da DTD e do XSD para imposição de restrições de integridade em documentos XML.

O capítulo 4 é dividido em duas seções. Na primeira são mostrados dois comparativos: o primeiro, entre as restrições de integridade do modelo relacional com os mecanismos de imposição de restrições de integridade da DTD; e o segundo, 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. 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, o primeiro dos trabalhos relacionados com relação às categorias de restrições de integridade de domínio do modelo relacional e o segundo dos trabalhos relacionados com relação à linguagem SQL. Esses comparativos estabelecem a motivação para a proposta da dissertação mostrada no capítulo 5.

Finalmente, no capítulo 5 são apresentados a arquitetura de validação de documentos XML, um estudo de caso e o cronograma com as atividades futuras.
2 Restrições de Integridade em Bancos de Dados Relacionais e SQL

Este capítulo apresenta o estudo feito sobre as restrições de integridade em bancos de dados relacionais e SQL\(^1\), seus conceitos, classificações e de que maneira elas podem ser definidas. A escolha desse modelo e dessa linguagem justifica-se pelo fato de apresentarem, respectivamente, uma teoria e prática de restrições de integridade.

2.1 Conceitos

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 a 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 ele se refere.

---

\(^1\) SQL (*Structured Query Language*) é a linguagem padrão para a definição e manipulação de bancos de dados relacionais.
2.2 Restrições de integridade em bancos de dados relacional

Existe uma grande variedade de restrições de integridade em bancos de dados relacional. 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-as como restrições de domínio, restrições de chaves, restrições de formas de relacionamentos e restrições de integridade referencial. ELMASRI & NAVATHE (2000) classificam as restrições da seguinte maneira: restrições de domínio, restrições de chave, restrições de entidade e restrições de integridade referencial. HEUSER (2001) classifica as restrições 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, podem ser classificadas as restrições de integridade em: restrições de domínio, restrições de chaves e restrições de integridade referencial, restrições quanto ao momento de verificação. Além destas, existe a classificação de restrições baseadas em eventos, que podem ser impostas através de uma chamada de aplicação ou de um evento no banco de dados. Essas classificações são vistas com mais detalhes a seguir.

2.2.1 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; não somente verificam os valores inseridos ou modificados no banco de dados, mas também testam as consultas para garantir que as 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 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: uma restrição 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 que é avaliada como sendo desse atributo deverá obedecer a essa restrição; em caso contrário, irá falhar.

- restrições de tipo: a restrição de tipo é apenas uma declaração para o efeito de que um atributo especificado seja de um tipo especificado, ou seja, as restrições de tipos 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 tuplas: é uma restrição sobre uma tupla individual. Podendo incorporar restrições sobre vários atributos da mesma. As restrições de tuplas 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 relaciona entre si duas ou mais tuplas distintas. Neste caso, é necessário que todas as restrições de tuplas envolvidas sejam atendidas; em caso contrário, a operação será desfeita. Por exemplo, o somatório das quantidades dos itens de um pedido não pode ser maior que o atributo quantidade total do pedido.

- restrições de transição de estado: as restrições 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, em um banco de dados de pessoas, haveria 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.
2.2.2 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 irreduzibilidade. A propriedade de unicidade é estabelecida quando nenhum valor válido de Y contém duas tuplas com o mesmo valor de X. A irreduzibilidade 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 tem de possuir um valor correspondente 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 da chave primária associada.

2.2.3 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 tem de existir.

2.2.4 Restrições quanto ao momento da verificação

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

Na imediata, a verificação é feita no momento da ocorrência de uma operação, ou seja, imediatamente. Já na postergada, as restrições são verificadas no momento que a instrução COMMIT\(^2\) é executada. Caso a restrição seja violada, uma instrução ROLLBACK\(^3\) é executada. Por exemplo, quando se necessita alterar um determinado código somando a ele um valor em todas as tuplas da relação. Essa operação só poderá ser efetivada no momento que todos os códigos 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 o ROLLBACK é executado.

2.2.5 Restrições baseadas em eventos

Existem restrições de integridade que podem ser programadas pelo usuário, e a sua verificação é independente das operações de atualização. Essas são denominadas restrições de integridade baseadas em eventos. Um exemplo é o pagamento automático de uma conta, ou seja, se o dia do vencimento da conta é igual ao dia atual, então a conta é paga, ou seja, as atualizações de saldo são feitas automaticamente no banco de dados.

---

\(^2\) COMMIT indica o término de uma operação bem-sucedida. Todas as operações feitas são gravadas.

\(^3\) ROLLBACK indica o término de uma operação mal-sucedida. Todas as operações serão desfeitas.
2.3 Restrições de integridade em SQL

SQL é a linguagem padrão para banco de dados relacionais. 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.

```sql
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 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 usada para assegurar que os valores dos atributos possam satisfazer a determinadas condições. A Figura 2.2 mostra um exemplo envolvendo os três tipos de restrições básicas de tabelas.

```sql
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 2.2- Exemplo de restrições básicas de tabelas.
- Restrições gerais: São definidas por meio da sintaxe CREATE ASSERTION. São restrições de declaração de predicados que devem ser sempre verdadeiros, geralmente envolvem várias tabelas do banco de dados e independentem da operação de atualização. SILBERCHATZ et al. (1999) definem como um predicado que expressa uma condição que deve ser sempre satisfeita no banco de dados. A Figura 2.3 mostra um exemplo de declaração de uma restrição geral, no qual toda pessoa tem idade maior que zero.

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

Fonte: Primária.

Figura 2.3- Exemplo de restrição geral.

- Gatilhos: Um gatilho é um conjunto de instruções que é executado automaticamente, em conseqüência da ocorrência de algum evento no banco de dados (SILBERCHATZ et al., 1999, DATE, 2000). São mecanismos úteis para avisos a usuários. 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 2.4, por exemplo, define um gatilho que tem a função de atualizar o atributo ATIVO na tabela de PESSOAS com o valor “S”. Esta ação ocorre após a inserção de um novo código.

| CREATE TRIGGER PESSOAS_AI 
| AFTER INSERT 
| BEGIN 
| Update PESSOAS 
| Set ATIVO='S' 
| Where CODIGO=new.codpes; |

Fonte: Primária.

Figura 2.4- Exemplo de gatilho.

- Procedimentos: Procedimentos são programas escritos em SQL que são armazenados em um 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 serem 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 da aplicação. Por exemplo, a Figura 2.5 mostra um procedimento que é executado toda vez que é processado um arquivo de pagamentos, efetuados pelos clientes, vindo do banco. 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;
```

Fonte: Primária.

Figura 2.5- Exemplo de procedimento.

- 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 aplicados 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 formam as transações podem ser embutidas em um programa da aplicação ou podem ser especificadas diretamente no banco de dados através de uma linguagem de consulta de alto nível, como SQL.

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,
supõe-se 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 um registro for adicionado a tabela SOCIOS_PESSOAS, tem-se de atualizar o atributo NUMERO_SOCIOS da tabela PESSOAS, ainda, toda vez que for excluído um registro 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 um registro 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;
```

Figura 2.6- Exemplo de transação.

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

A restrição NOT DEFERRABLE é sempre verificada imediatamente. A restrição DEFFERABLE pode ser definida como INITIALLY DEFERRED (estado em que a restrição não é verificada) e INITIALLY IMMEDIATE (estado em que a
resolução é verificada). Esses valores que definem o estado de início da transação e podem ser ativados e desativados dinamicamente por meio da instrução SET CONSTRAINTS <NOME_RESTRIÇÃO> <OPÇÃO>, onde <OPÇÃO> tem os valores DEFERRED ou IMMEDIATE.

2.4 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 2.1 mostra este comparativo.

<table>
<thead>
<tr>
<th>Tabela 2.1 - Comparativo relacional vs. SQL.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SQL</strong></td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td><strong>Relacional</strong></td>
</tr>
<tr>
<td>a) Domínio</td>
</tr>
<tr>
<td>Atributos</td>
</tr>
<tr>
<td>Tipo</td>
</tr>
<tr>
<td>Tuplas</td>
</tr>
<tr>
<td>Banco de dados</td>
</tr>
<tr>
<td>Transição de estado</td>
</tr>
<tr>
<td>b) Chaves</td>
</tr>
<tr>
<td>candidatas</td>
</tr>
<tr>
<td>primárias</td>
</tr>
<tr>
<td>estrangeiras</td>
</tr>
<tr>
<td>c) Integridade referencial</td>
</tr>
<tr>
<td>d) Momento de Verificação</td>
</tr>
<tr>
<td>e) Baseada em eventos</td>
</tr>
</tbody>
</table>
Analisando a Tabela 2.1, verifica-se que as restrições de domínio do modelo relacional são na sua totalidade atendidas de alguma maneira pela linguagem SQL. As restrições de domínio, por exemplo, são principalmente impostas através do uso da cláusula CHECK, gatilhos e procedimentos. Também se verifica que as restrições de chaves são impostas na criação de tabelas através das cláusulas PRIMARY KEY, FOREIGN KEY e UNIQUE.

As restrições quanto ao momento de verificação na linguagem SQL são todas do tipo imediato, exceto no corpo de uma transação onde podem ser declaradas como postergadas. Já as restrições baseadas em eventos podem ser declaradas na linguagem SQL através do uso de procedimentos (chamada de uma aplicação) e também por gatilhos (disparado automaticamente pelo sistema gerenciador de banco de dados).

### 2.5 Considerações finais

Neste capítulo foi realizado um estudo das restrições de integridade do modelo relacional e da linguagem SQL. A escolha deste modelo e desta linguagem se justifica por apresentarem, respectivamente, teoria e prática consolidadas para o tratamento de restrições de integridade em bancos de dados relacionais.

Através desse estudo, verificou-se por meio da análise das obras de vários autores, que existem várias classificações de restrições de integridade e sentiu-se a necessidade de estabelecer uma classificação única para as restrições de integridade e a partir dela efetuar uma análise dos recursos que a linguagem SQL disponibiliza para seus tratamentos. Nessa análise, conclui-se que a linguagem SQL possui recursos para tratar todas as classificações de restrições de integridade do modelo relacional que foram estabelecidas pela classificação.

Além disso, constata-se, que na classificação de restrições de integridade de domínio, as restrições são impostas pela linguagem SQL principalmente através de gatilhos, procedimentos, checks e asserções.
3 XML

Este capítulo, primeiramente, dá uma visão geral da tecnologia XML. Em seguida, aborda dois padrões de manipulação da tecnologia XML (DTD e XSD), dando uma ênfase nos tópicos que se referem a restrições de integridade.

3.1 Tecnologia XML

De acordo com o W3C, 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 define uma maneira padrão para delimitar dados de textos. Segundo ANDERSON et al.(2000), XML é “o ASCII para a WEB”. Com ele é possível, através de uma determinada linguagem de programação, criar uma estrutura de dados arbitrária e, então, compartilhá-la com outras linguagens em outras plataformas computacionais; pode-se, assim, descrever formalmente a sintaxe que se deseja conceber e, então, compartilhar.

3.2 Dados XML e Documentos XML

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. Utiliza delimitadores chamados tags para descrever seus dados. Uma tag indica a intenção do dado e delimita o seu conteúdo. Um dado em XML

---

4 W3C (World Wide Web Consortium) é um consórcio formado por acadêmicos e empresários que buscam definições de padrões para WEB.
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 em um documento XML. A Figura 3.1 mostra um exemplo de documento XML para o domínio de cadastro de pessoas.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<pessoas>
  <codigo id="C1">
    <nome>
      <primeiro> Maria </primeiro>
      <meio> Aparecida </meio>
      <sobrenome> Tavares </sobrenome>
    </nome>
    <endereco>
      <rua> Rua das Flores </rua>
      <numero> 1789 </numero>
      <bairro> Centro </bairro>
    </endereco>
  </codigo>
</pessoas>
```

Figura 3.1- Exemplo de um documento XML

A organização dos dados em um documento XML é hierárquica, conforme mostra a Figura 3.1, 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`, `endereco`, `cidade`, que, neste caso, são filhos do elemento `codigo`, que, por sua vez, é filho do elemento `pessoa`.

Os elementos podem conter atributos, que são informações que se deseja anexar sobre o elemento em questão. ANDERSON et al. definem: “Se elementos são nomes de XML, então atributos são seus adjetivos”. Por exemplo, na Figura 3.1 tem-se o atributo `id` especificando uma especialização do 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*. Um documento *bem formado* é composto de três partes: (i) prólogo, que é o sinal do início do documento, onde
descreve dicas de configurações do mesmo; (ii) corpo, que consiste de um ou mais elementos organizados em forma de árvore hierárquica; e (iii) epílogo, onde se tem comentários e instruções de processamento.

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 hierárquico (DTD ou XSD) seja fornecido pelo documento.

A outra forma de restrição de documentos XML refere-se a um documento *válido*, que é dito válido se possui um esquema hierárquico definido (DTD ou XSD) e se a sua estrutura está de acordo com este esquema. Os esquemas hierárquicos permitem que um determinado vocabulário XML possa comunicar as regras sintáticas e estruturas com outro vocabulário XML. A seção 3.3 (Esquemas XML) mostra com mais detalhes as estruturas hierárquicas.

A validação, se um documento XML é *bem formado* e/ou se um documento XML é *válido*, é feita pelos *parsers*, que são programas com instruções para efetuar as validações de documentos XML. Em adição à especificação da sintaxe XML, o W3C descreveu alguns comportamentos da arquitetura cliente XML, o *XML processor* ou *parser*, no caso da necessidade de desenvolvimento. Existem dois tipos de *parsers*: um que não efetua a validação do documento, verificando apenas se o documento é *bem formado*, chamado de *non-validating*, e o outro, chamado de *validating*, que efetua a validação do documento caso seja *bem formado*.

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 elas, DTD e XSD são vistas a seguir.

### 3.3 Esquemas XML

Segundo a definição do W3C, “esquemas XML expressam vocabulários compartilhados, que permitem as máquinas carregar regras feitas por pessoas. Eles
fornecem uma maneira de definição da estrutura, conteúdo e semântica de documentos XML”.(W3CXMSchema, 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.

3.3.1 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.”

A DTD descreve tudo que pode estar contido em um documento XML. Descrever o que cada elemento e atributo faz torna a DTD mais legível. Portanto, ela realiza uma série de tarefas, como definir e fornecer todos os nomes de todos os elementos utilizados no documento; definir e fornecer todos os nomes dos atributos usados no documento; definir e fornecer o nome de todas as entidades usadas no documento; especificar a ordem em que os elementos e atributos devem aparecer no documento; definir os atributos e seus valores padrão de cada elemento e descrever comentários que possam esclarecer o contexto estrutural do documento.

As declarações são feitas através de palavras-chaves. A Tabela 3.1 mostra os tipos de declarações, com suas palavras-chaves e significados, que podem ser possíveis de ocorrer em uma DTD.

<table>
<thead>
<tr>
<th>Palavra-Chave</th>
<th>Significado</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTLIST</td>
<td>Declaração de atributos, que podem especificar um tipo de elemento e os valores permitidos para o atributo.</td>
</tr>
<tr>
<td>ELEMENT</td>
<td>Declaração de elementos.</td>
</tr>
<tr>
<td>ENTITY</td>
<td>Declaração de reuso de um conteúdo.</td>
</tr>
<tr>
<td>NOTATION</td>
<td>Formato de declaração para conteúdo externo.</td>
</tr>
</tbody>
</table>
No padrão DTD, os elementos são declarados através da palavra-chave <!ELEMENT, seguida pelo seu nome e tipo. Por exemplo, na Figura 3.2, as declarações dos elementos: pessoa, código, nome, endereço, cidade, primeiro, meio, sobrenome, rua, número e bairro.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE pessoas [  
<!ELEMENT pessoa (código+)>
<!ELEMENT código (nome, endereço, cidade)'>
<!ATTLIST código id CDATA>
<!ELEMENT nome (primeiro, meio?, sobrenome)>
<!ELEMENT primeiro (#PCDATA)>
<!ELEMENT meio (#PCDATA)>
<!ELEMENT sobrenome (#PCDATA)>
<!ELEMENT endereço (rua, número, bairro)>
<!ELEMENT rua (#PCDATA)>
<!ELEMENT número (#PCDATA)>
<!ELEMENT bairro (#PCDATA)>
<!ELEMENT cidade (#PCDATA)>
]>
```

Fonte: Primária.

Figura 3.2- 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 3.2, os elementos primeiro, meio, sobrenome, rua, bairro e número são elementos simples.

Elementos complexos são elementos que contêm outros elementos, como mostra, na Figura 3.2, as declarações dos elementos pessoa, código, nome e endereço.

Ao declarar elementos, podem-se definir controles de conteúdo, de sequência 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, podem-se colocar dados caracteres, elementos e até mesmo referências a informações externas. 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*) >`

Os controles de sequência 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 sequência dos elementos-filhos e dos conteúdos, além de definir opção entre elementos e conteúdos. Os controles de sequência 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.

Controles de cardinalidade\(^5\) oferecem mecanismos para controlar a frequência 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 3.2;

\(^5\) Cardinalidade é a quantidade de ocorrências que o elemento pode ter num documento XML.
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 3.2;

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 3.2, a declaração do elemento NOME.

Um elemento que não possui indicador de ocorrência 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 código 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 3.2 mostra a classificação dos tipos de atributos com significados e exemplos.

A declaração do padrão do atributo define a maneira em que ele aparece no documento e pode conter três tipos de valores: #REQUIRED, #IMPLIED e #FIXED. A Tabela 3.3 mostra cada uma das declarações de padrões e seus respectivos significados.

No padrão DTD existe a possibilidade de definir entidades. Segundo PITTS-MOULTIS & KIRK (2000), “entidades são aqueles grupos de marcação que efetivamente possuem conteúdo no interior de suas definições”. Declara-se uma entidade na DTD; depois, refere-se a ela através de uma referência no documento XML, ou seja, é um conteúdo associado a um nome.

A declaração de uma entidade é feita através da palavra-chave ENTITY. XML suporta dois tipos de entidades: genéricas ou gerais e paramétricas ou parâmetros. As entidades genéricas ou gerais especificam caracteres reservados e especializados,
começam com o ‘&’ e terminam com o ‘;’. Entidades gerais são usadas para definir um texto que será inserido no conteúdo do documento, mas não podem ser usadas para trabalhar com declarações propriamente ditas na DTD. Por exemplo, para declarar utiliza-se <! ENTITY xml “Extensible Markup Language” > e, para referenciar, utiliza-se &xml;.

Tabela 3.2- Tipos de atributos.

<table>
<thead>
<tr>
<th>Tipo</th>
<th>Significado</th>
</tr>
</thead>
<tbody>
<tr>
<td>CDATA</td>
<td>Define um atributo do tipo texto (string).</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST pessoal cargo CDATA #IMPLIED &gt;</td>
</tr>
<tr>
<td>ID</td>
<td>O atributo é um nome único dentro do documento.</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST pessoa cpf ID #REQUIRED &gt;</td>
</tr>
<tr>
<td>IDREF</td>
<td>O atributo é uma referência a um atributo do tipo ID com o mesmo valor.</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST pessoa cidade IDREF #IMPLIED &gt;</td>
</tr>
<tr>
<td>IDREFS</td>
<td>Referência a uma série de IDs, separados por espaços.</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST pessoa cidades IDREFS #IMPLIED &gt;</td>
</tr>
<tr>
<td>ENTITY</td>
<td>Referência a uma entidade externa predefinida.</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST pessoa foto ENTITY #IMPLIED &gt;</td>
</tr>
<tr>
<td>ENTITIES</td>
<td>Referência a uma série de entidades, separados por espaços.</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST pessoa fotos ENTITIES #IMPLIED &gt;</td>
</tr>
<tr>
<td>NM_TOKEN</td>
<td>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. Não pode incluir espaços em branco. O valor consiste de uma única palavra.</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST cidade uf NM_TOKEN #REQUIRED &gt;</td>
</tr>
<tr>
<td>NM_TOKENS</td>
<td>São atributos que possuem valores compostos de vários tokens, cada um deve ser um token válido, e são separados por espaços.</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST pessoa senhas NM_TOKENS #REQUIRED &gt;</td>
</tr>
<tr>
<td>NOTATION</td>
<td>Usados para referenciar dados externos ligados ao documento XML ou definir um valor para o atributo.</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST jpg SYSTEM “jpgviewer.exe” &gt;</td>
</tr>
<tr>
<td></td>
<td>&lt;! ATTLIST imagem tipoimagem NOTATION (gif</td>
</tr>
<tr>
<td>[valores definidos]</td>
<td>Define um conjunto de valores que podem ser definidos para o atributo.</td>
</tr>
<tr>
<td></td>
<td>Ex: &lt;! ATTLIST pessoal ativo (sim</td>
</tr>
</tbody>
</table>

Entidades paramétricas ou parâmetros são criadas como um nome alternativo para um grupo de elementos, normalmente conteúdo de elementos e atributos, que são usados frequentemente na DTD. Essas entidades são usadas dentro das DTDs para melhorar a organização e a eficiência. Referências a entidades parâmetros começam com ‘%’ e terminam com o ‘;’. Por exemplo, a declaração <! ENTITY secao (#CDATA | subgroup) > e a referência %secao;

Uma entidade possui seu conteúdo associado a um nome. Este nome pode estar contido na declaração da entidade (entidade interna) ou fora da declaração (entidade externa). Por exemplo, <! ENTITY %pessoas SYSTEM “/dtds/pessoas.dtd” >
Tabela 3.3- Padrões de atributos.

<table>
<thead>
<tr>
<th>Padrões</th>
<th>Significado</th>
</tr>
</thead>
<tbody>
<tr>
<td># REQUIRED</td>
<td>É obrigatória a presença do atributo em toda instância do elemento.</td>
</tr>
<tr>
<td># IMPLIED</td>
<td>O atributo pode opcionalmente aparecer.</td>
</tr>
<tr>
<td># FIXED</td>
<td>O valor é fixo, não pode ser modificado.</td>
</tr>
</tbody>
</table>

O conteúdo de uma entidade pode ser do tipo texto ou binário. Nas entidades do tipo texto, os dados do tipo caractere são atribuídos como seu conteúdo e, quando ela é referenciada, o conteúdo da entidade é inserido no documento no lugar da referência à entidade. Elas são especialmente usadas na criação de nomes alternativos para frases e blocos de texto ou de marcação usados com freqüência, bem como para a inserção de caracteres especiais. As entidades binárias caracterizam-se por um recurso que não é codificado em XML. Por exemplo, um arquivo de vídeo é considerado uma entidade binária.

A especificação XML possui cinco entidades predefinidas nativas. Para seu uso não é necessária a definição delas em subconjuntos de DTDs externas nem internas. Essas entidades são mostradas na Tabela 3.4.

Tabela 3.4- Entidades predefinidas.

<table>
<thead>
<tr>
<th>Carácter</th>
<th>Entidade</th>
<th>Significado</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>Sinal de maior que</td>
</tr>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>Sinal de menor que</td>
</tr>
<tr>
<td>&amp;</td>
<td>&amp;</td>
<td>E-comercial</td>
</tr>
<tr>
<td>'</td>
<td>'</td>
<td>Apóstrofo</td>
</tr>
<tr>
<td>“</td>
<td>&quot;</td>
<td>Aspas</td>
</tr>
</tbody>
</table>

Outro recurso da tecnologia XML que se pode utilizar em DTDs é a declaração de notações. O objetivo de uma notação é ajudar o aplicativo que irá processar o documento XML a trabalhar com entidades não-XML. Para declarar uma notação utiliza-se a palavra-chave NOTATION. Por exemplo, `<! NOTATION doc PUBLIC "c:\word\winword.exe" >.`

Os comentários em XML nada mais são do que notas em um documento XML ignorado pelo processador XML. Para definir um comentário é necessário que o texto esteja entre `<!--' e '-->'`. Por exemplo, `<!-- Isto é um comentário em XML -->.`
3.3.2 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 DOM⁶;

- 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*⁷, 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, conteúdo e semântica de documentos XML, conforme um modelo definido em um padrão XML *Schema* (ANDERSON *et al*., 2000, W3C, BRADLEY, 2004).

Um esquema XSD, conforme mostra a Figura 3.3, 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.

---

⁶ DOM é a interface de programação padronizada pelo W3C para tratar documentos XML.

⁷ *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.
Os elementos podem ter um prefixo xsd, o qual está associado com um namespace declarado no elemento schema, conforme mostra a Figura 3.3. O propósito dessa associação é identificar os elementos e os tipos simples pertencentes a um determinado vocabulário.

![XML Schema](xmlschema.png)

Fonte: Primária.

Figura 3.3- Exemplo de XSD.

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 irá conter, declaração feita através da palavra type. Por exemplo, na Figura 3.3, a declaração do elemento PRIMEIRO.
Similmente 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 3.3.

A declaração de um atributo pode ocorrer diretamente dentro do elemento **schema**. Neste caso, ele poderá ser compartilhado por todas as declarações de elementos. Em outro caso, a declaração do atributo pode estar embutida dentro de um determinado elemento; assim, fica amarrado ao elemento junto ao qual foi declarado. Por exemplo, a declaração `<attribute name="primeiro" type="string" />`. Ao associar essa declaração com o elemento **nome**, tem-se o seguinte exemplo válido: `<nome primeiro="Alexandre"> ... </nome>.

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. O conteúdo, por exemplo, pode ser uma mistura de texto e elementos. As declarações podem ser feitas das seguintes formas: `<element name="observacao" />, ou `<element name="observacao" type="anyType" />.`

- **elementos vazios**: normalmente, fornecem instruções que criam ou referenciam outras declarações de elementos dentro do elemento corrente. Para definir um elemento com conteúdo vazio é necessário definir um tipo que permita apenas elementos em seu conteúdo, porém eles não podem ser declarados. É necessária a declaração do elemento **complexType**. Por exemplo, a Figura 3.4 mostra a declaração de um elemento vazio.

```xml
<element name = "comentario">
  <complexType>
    </complexType>
</element>
```

Fonte: Primária.

Figura 3.4- Exemplo de elemento com conteúdo vazio.
• conteúdo simples: um elemento com conteúdo do tipo simples ocorre quando o estado do seu conteúdo está de acordo com um tipo de dados. O atributo type é adicionado ao elemento para esse propósito. Por exemplo, a declaração do elemento PRIMEIRO na Figura 3.3.

• 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 3.5 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>
```

Figura 3.5- 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: a declaração de um elemento contém a especificação do conteúdo permitido, o qual inclui 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 3.3 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. Também precisa declarar como esses elementos serão organizados. Eles podem aparecer em uma ordem predefinida, não ter restrições de ordem, ou consistir numa série de alternativas. Por exemplo, a declaração do elemento NOME na Figura 3.3.

• elementos sequência: quando elementos filhos necessitam ocorrer em uma determinada ordem na instância de um documento, utiliza-se o elemento sequence. Por exemplo, na Figura 3.3 a declaração do elemento NOME.

• elementos opcionais e repetitivos: por padrão, uma referência a um elemento indica que ele deve estar presente na instância do documento, porém existem casos em que a sua presença não é obrigatória, e, sim opcional. 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 3.3 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. Porém, através da definição do atributo minOccurs=0 para cada elemento, não se tem a garantia de que ambos, ou nenhum dos dois, possam ser referidos. Para garantir que apenas um elemento seja referenciado no documento, utiliza-se o elemento choice. Por exemplo, na Figura 3.3 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 3.6 mostra um exemplo de definição do elemento all. Este tipo de grupo não pode ser empregado dentro de outros grupos de elementos nem pode conter elementos do tipo any.

```xml
<all>
  <element ref="CPF"/>
  <element ref="RG"/>
  <element ref="PIS" minOccurs="0"/>
</all>
```
Fonte: Primária.

Figura 3.6- 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, e também dentro de outra instância do mesmo elemento. Através dessas combinações podem-se criar modelos complexos. Na Figura 3.3 a declaração de elemento identificação 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. O elemento any pode ser declarado dentro de um elemento sequence ou dentro de um elemento choice e pode também ter um atributo namespace, que identifica os elementos permitidos através do nome do namespace, ou namespaces, a que eles devem pertencer. Um exemplo é mostrado na Figura 3.7.

```xml
<element name="fotos">
  <complexType>
    <sequence>
      <any minOccurs="0"
           maxOccurs="unbounded"
           namespace="#local"/>
    </sequence>
  </complexType>
</element>
```
Fonte: Primária.

Figura 3.7 - Exemplo do elemento any.

• grupos compartilhados: quando dois ou mais elementos possuem conteúdos de estruturas idênticas, eles não necessitam ser 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. A definição do grupo é referenciada, sempre que necessário, através de outro elemento group que é vazio, mas inclui um atributo ref que referencia o nome do grupo que definiu a estrutura. Por exemplo, a Figura 3.8 mostra a definição de um grupo compartilhado.

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

Fonte: Primária.

Figura 3.8- Exemplo do elemento group.

A declaração de atributos, similarmente à declaração de elementos, pode conter especificação do tipo de conteúdo. A Figura 3.9 mostra alguns exemplos dessas declarações.

Segundo BRADLEY (2004), existem várias maneiras de anexar atributos a uma declaração de elementos. Pode-se ter atributos em elementos vazios, nesse caso, o elemento é considerado complexo, como mostra na Figura 3.9 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" / >.

Pode-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 3.9, a declaração do atributo complemento no elemento endereco.

A outra 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, referenciando o tipo de dados que será estendido. Por exemplo, na Figura 3.9 a declaração do atributo UF no elemento CIDADE.

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

Fonte: Primária.

Figura 3.9- Exemplos de declarações de atributos.

A existência de um atributo na instância de um documento é, por padrão, opcional, entretanto sua presença pode ser obrigatória ou proibida. O atributo use é utilizado para especificar essas declarações. Ele pode conter um dos valores: optional, required e prohibited. Por exemplo, `<attribute name = "id" use = "required" ... />`. 
Um atributo pode ter um valor padrão (*default*). Esse valor 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 3.9 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 3.9 a declaração do tipo da dados ESTADOS e a sua referência na declaração do atributo UF.

Através da definição do tipo de dados podem-se impor restrições ao conteúdo de um elemento ou atributo específico. Tipos de dados são geralmente baseados em outros tipos de dados similares, porém menos restritivos, assim formando uma hierarquia de tipos. No topo desta hierarquia está o tipo *anyType*, que se subdivide em *anySimpleType* (tipos simples) e *all complex types* (tipos complexos). Os tipos de dados simples são direta ou indiretamente baseados no tipo *anyType*; já os complexos envolvem atributos e elementos filhos.

O padrão XML Schema inclui uma variedade nativa de tipos de dado simples criada a partir do tipo *anySimpleType*, chamados de tipos primitivos de dados (*primitive data types*).

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 3.9 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. Essas restrições são conhecidas como *facets*. A
Tabela 3.5 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 3.5- Tipos de *facets*.

<table>
<thead>
<tr>
<th><em>facets</em></th>
<th>Definição</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td>Restringe o valor para um número fixo de caracteres.</td>
</tr>
<tr>
<td>Minimum length</td>
<td>Restringe o valor para um número mínimo de caracteres.</td>
</tr>
<tr>
<td>Maximum length</td>
<td>Restringe o valor para um número máximo de caracteres.</td>
</tr>
<tr>
<td>Pattern</td>
<td>Define um template contra o qual o valor deve ser comparado.</td>
</tr>
<tr>
<td>enumeration</td>
<td>Define as possibilidades de valores.</td>
</tr>
<tr>
<td>whitespace</td>
<td>Define a maneira que os caracteres de espaço (alimentação de linha, espaço em branco, tabulação, ..) aparecem. Pode ter os valores “replace”, “collapse” e “preserve”.</td>
</tr>
</tbody>
</table>

Tipos simples que possuem valores numéricos têm *facets* adicionais que são chamadas “*facets* enumeradas”; elas 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 3.6 mostra os tipos de *facets* enumeradas e as suas definições e o Anexo 2, uma associação dos tipos de *facets* enumeradas permitidas com os tipos simples de dados.

### Tabela 3.6- Tipos de *facets* enumeradas.

<table>
<thead>
<tr>
<th><em>facets</em></th>
<th>Definição</th>
</tr>
</thead>
<tbody>
<tr>
<td>maximum inclusive</td>
<td>Especifica o valor máximo, incluindo o próprio valor.</td>
</tr>
<tr>
<td>maximum exclusive</td>
<td>Especifica o valor máximo, excluindo o próprio valor.</td>
</tr>
<tr>
<td>Minimum inclusive</td>
<td>Especifica o valor mínimo, incluindo o próprio valor.</td>
</tr>
<tr>
<td>Minimum exclusive</td>
<td>Especifica o valor mínimo, excluindo o próprio valor.</td>
</tr>
<tr>
<td>total digits</td>
<td>Especifica o número máximo de dígitos permitidos no valor numérico.</td>
</tr>
<tr>
<td>fractional digits</td>
<td>Especifica o número de dígitos que seguem o ponto decimal.</td>
</tr>
</tbody>
</table>

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. A Tabela 3.7 mostra esses elementos com exemplos e comentários.

Uma maneira de estender um tipo de dado simples é através da criação de listas de valores. O elemento *list* cria uma lista de valores de um determinado tipo; o atributo *itemType* define o tipo de dado dos itens da lista; os itens que compõem a lista são separados por espaços. Para impor restrições nas listas podem-se utilizar as seguintes *facets*: length, minun length, maximun length, pattern e enumeration. Como
Exemplo, a Figura 3.10 mostra uma definição de listas onde o tipo de dados é inteiro e o tamanho da lista é 10.

<table>
<thead>
<tr>
<th>Facets</th>
<th>Exemplos</th>
<th>Comentários</th>
</tr>
</thead>
</table>
| Length            | `<simpleType name="strcpf">  
  <restriction base="string">  
  <length value="11"/>  
  </restriction>
</simpleType>` | Define que o número de caracteres dos elementos que referenciam o tipo "strcpf" é 11. |
| minLength, maxLength | `<simpleType name="strcpfgc">  
  <restriction base="string">  
  <minlength value="11"/>  
  <maxlength value="14"/>  
  </restriction>
</simpleType>` | Define que o número mínimo de caracteres dos elementos que referenciam o tipo "strcpfgc" é 11 e o máximo é 14. |
| Pattern           | `<simpleType name="masvalor">  
  <restriction base="string">  
  <pattern value="\d\d\d\d\d\d(4)"/>  
  </restriction>
</simpleType>` | 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"/>  
  <enumeration 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>
</simpleType>` | 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>
</simpleType>` | 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>
</simpleType>` | Define que os valores válidos dos elementos devem ter um total de 4 dígitos sendo que 2 devem ser na parte fracionária. |

É possível criar um novo tipo de dados através de uma união de dois ou mais tipos de dados existentes. Muitos tipos de dados utilizados são predefinidos na type library fornecida com o padrão XML Schema. Os valores para o novo tipo são considerados
válidos se estão conformes às restrições de qualquer um dos tipos. O elemento union e o atributo memberTypes são utilizados na definição. Por exemplo, a união dos tipos time e gMonthDay, <union memberTypes="time getMonthDay" />

```
<simpleType name="listahorarios">
  <list itemType="integer"/>
</simpleType>
<simpleType name="Tamlistahorarios">
  <restriction base="listahorarios"/> <length value="10"/> </restriction>
</simpleType>
```

Fonte: Primária.

Figura 3.10- Exemplo de definição de listas.

Um elemento que requer atributos ou possui um conteúdo que querer elementos filhos deve ser criado com a ajuda de uma definição de “tipo complexo”. Segundo BRADLEY (2004), os tipos complexos podem ser: (i) uma variante estendida de um tipo complexo de dados, adicionando elementos e atributos; (ii) uma variante restrita de um tipo complexo de dados, removendo ou restringindo o conteúdo de elementos e atributos; (iii) uma variante restrita de um tipo complexo de dados que têm um modelo de conteúdo simples; (iv) uma variante estendida de um tipo simples de dados, adicionando atributos ao elemento simples.

Um definição de tipo complexo pode ser compartilhada se o elemento complexType é colocado diretamente dentro do elemento schema. O atributo name dá o nome ao novo tipo complexo e deve ser único entre todos os nomes de tipos de dados, porém pode ser o mesmo de uma outra construção, tal como nome de elementos ou definição de grupo de atributos. Ele pode ser referenciado a partir de um elemento, da mesma maneira que um tipo simples, ou através de um outro tipo complexo, ou por um outro esquema através do uso de namespaces.

Um tipo de dados complexo pode ser derivado por extensão ou restrição de um outro tipo complexo, embora não seja necessário que seja derivado de outro tipo complexo. O elemento complexContent especifica que um tipo complexo é derivado de outro tipo complexo.

O elemento complexContent pode conter os elementos extension e restriction. O elemento extension indica uma derivação por extensão e que, portanto, novos
componentes serão adicionados ao modelo existente; o atributo **base** indica o tipo de dados existente que está sendo estendido. Na Figura 3.10, a declaração do elemento **cidade** mostra um exemplo de derivação por extensão.

Na derivação por restrição, um tipo complexo de dados é uma variante de outro tipo complexo de dados. Primeiramente, tem-se a replicação do modelo; depois, são impostas restrições a ele. Por exemplo, na Figura 3.10 a declaração do tipo **estados**.

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. O termo fragmento único (**unique fragment**) é usado para descrever qualquer elemento que possui existência própria e que contém todas informações necessárias para ser identificado e localizado por seleção ou extração. Tal fragmento pode ser de qualquer tamanho, ser um elemento vazio ou um elemento que contém texto ou outros elementos.

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

Quando uma DTD é utilizada numa instância do documento e se deseja que determinados atributos sejam únicos no documento, eles devem ser definidos com o tipo **ID**; a referência a esses atributos é feita através da declaração do tipo **IDREF**.
A utilização de DTDs em documentos XML, para identificação de chaves e referências, é limitada devido a algumas situações: a chave deve ter uma única palavra; a chave deve ser um valor simples, não podendo ser uma combinação de valores; a chave é limitada a um pequeno número de caracteres válidos; a chave deve ser um valor de atributo, não podendo ser um elemento; o fragmento identificador deve ser o elemento que contém o atributo que é o valor chave; apenas uma série de valores únicos é permitida na instância do documento, pelo perigo de uma duplicação inesperada de chaves, quando validações de unicidade são feitas em separado em várias seções não relacionadas do documento. O XML Schema fornece características mais sofisticadas para tratar essas limitações.

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. Em casos simples, é inserido na definição do elemento raiz; em outros casos, o elemento escolhido deve ser um elemento complexo, porque não é possível a identificação de valores de fragmentos em tipos de dados simples. A declaração deve, então, referir-se a um tipo complexo de dados, através do atributo `type`, ou conter um elemento `complexType`. Em último caso, o elemento `unique` é colocado após o elemento `complexType`. O atributo `name` dá um nome que identifica o fragmento único. A Figura 3.11 (a) mostra um exemplo de definição do elemento `unique`.

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 3.11 (b) mostra um exemplo onde elemento `unique` na Figura 3.11 (a) pode ser substituído pelo elemento `key`.

Ao utilizar os elementos `unique` ou `key`, duas tarefas devem ser realizadas: a primeira, os fragmentos devem ser identificados, por exemplo, o nome de cada pessoa; a segunda, o valor ou valores desses fragmentos que servirão como chave deverão ser isolados, por exemplo, o valor do CPF de cada nome de pessoa (BRADLEY, 2004).
Figura 3.11- Exemplo de referências de chaves.

O elemento selector realiza a tarefa de encontrar fragmentos. Ele pode ser precedido de um elemento annotation, que é utilizado para conter elementos documentation (comentários para seres humanos) e appinfo (instruções para a máquina interpretadora XML); deve, entretanto, ser o primeiro elemento dentro do elemento unique ou key. 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 3.8 mostra os possíveis valores e seu significado.

<table>
<thead>
<tr>
<th>Expressão Xpath</th>
<th>Significado</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>*</code></td>
<td>Todos elementos filhos.</td>
</tr>
<tr>
<td><code>A</code></td>
<td>Elementos filhos de A.</td>
</tr>
<tr>
<td><code>child::A</code></td>
<td>Elementos filhos de A.</td>
</tr>
<tr>
<td><code>A/B</code></td>
<td>Elementos filhos B dos elementos filhos A.</td>
</tr>
<tr>
<td>`A</td>
<td>B`</td>
</tr>
<tr>
<td>`A/B</td>
<td>C/D`</td>
</tr>
<tr>
<td><code>.//</code></td>
<td>Todos elementos descendentes.</td>
</tr>
<tr>
<td><code>.//B</code></td>
<td>Todos elementos descendentes de B.</td>
</tr>
<tr>
<td><code>A // B</code></td>
<td>Todos elementos descendentes de B com todos elementos filho de A.</td>
</tr>
</tbody>
</table>

O elemento field é usado nos elementos unique e key para encontrar os valores que serão chave 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.

Uma instância de um documento pode incluir uma série de referências (key holders) a fragmentos que contêm chaves, que podem ser úteis para validar cada referência à chave (key references), porque uma referência pode estar correta, mas o fragmento a que se refere pode não existir; ou a referência pode estar correta, mas o fragmento a que se refere tem uma chave incorreta; ou a referência pode, simplesmente, não incluir uma cópia exata da chave no fragmento supostamente descrito.

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 3.12 (a) mostra um exemplo de definição de um XSD utilizando o elemento keyref.

3.4 Considerações finais

Neste capítulo, além de mostrar o crescimento da utilização e a importância de documentos XML para troca de dados na WEB, realizou-se um estudo com o objetivo de levantar quais 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, feitas através dos elementos key, unique e keyref.
4 Restrições de Integridade para Dados XML

Este capítulo mostra, primeiramente, um estudo comparativo das restrições de integridade em bancos de dados relacionais, mostrado no capítulo 2, com a tecnologia XML, mais especificamente em esquemas DTD e XSD, mostrados no capítulo 3. O objetivo deste capítulo é verificar como esquemas XML tratam essas restrições de integridade e as deficiências desse tratamento. Em seguida, são apresentados e analisados alguns trabalhos existentes referentes a restrições de integridade de domínio que serão o foco da dissertação, como explicado no capítulo 5.

4.1 Relacional vs. Esquemas 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, devido a suas características (representação auto-descritiva, 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 de restrição de integridade de tipo e restrição de integridade de atributo.

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 seguir, nas seções 4.1.1 e 4.1.2, são mostrados os estudos comparativos entre esquemas DTD e XSD com as classificações de restrições de integridade do modelo relacional.

4.1.1 Relacional vs. DTD

Uma DTD, no que se refere à imposição de restrições de integridade, possui controles de conteúdo, controles de seqüência e controles de cardinalidade, além de especificar o tipo de elemento e a ocorrência dos atributos na sua declaração. A Tabela 4.1 mostra uma associação desses controles com a classificação de restrições de integridade do modelo relacional.

<table>
<thead>
<tr>
<th>Tabela 4.1- Comparativo DTD vs relacional.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DTD</strong></td>
</tr>
<tr>
<td>a) Domínio</td>
</tr>
<tr>
<td>a) Domínio</td>
</tr>
<tr>
<td>a) Domínio</td>
</tr>
<tr>
<td>a) Domínio</td>
</tr>
<tr>
<td>b) Chaves</td>
</tr>
<tr>
<td>b) Chaves</td>
</tr>
<tr>
<td>b) Chaves</td>
</tr>
<tr>
<td>b) Chaves</td>
</tr>
<tr>
<td>c) Integridade referencial</td>
</tr>
<tr>
<td>d) Momento de Verificação</td>
</tr>
<tr>
<td>e) Baseada em eventos</td>
</tr>
</tbody>
</table>

Analisando a Tabela 4.1, verifica-se que a maior quantidade 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.

Os controles de sequência (VÍRGULA, BARRA VERTICAL) também podem ser classificados como uma restrição de tipo, pois são utilizados para determinar a estrutura dos elementos, através da sequência 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 estão descrevendo quais são os elementos que os compõem e restringindo as suas quantidades de ocorrência, ou seja, definindo a sua estrutura.

Os controles de tipos de atributos (CDATA, ID, IDREF, IDREFS, ENTITY, ENTITIES, NM_TOKEN, NM_TOKENS, 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, SAX⁹ (Simple API for XML) e DOM¹⁰ (Document Object Model), com uma linguagem de programação e um parser (código de verificação) que oferece suporte a essa linguagem, porém elas 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).

---

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

Através da Tabela 4.1 nota-se, na especificação da DTD, a ausência de controles para imposição de restrições de integridade de tuplas e de banco de dados semelhantes as existentes no modelo relacional.

4.1.2 Relacional vs. XSD

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

Esses controles podem ser classificados como conteúdo de elementos, sequência e escolha de elementos, padrões de elementos, ocorrência de elementos, conteúdo de atributos, padrões de atributos, ocorrência de atributos, tipos de dados e chaves e referências. As Tabelas 4.2 (I) e 4.2 (II) mostram um comparativo desses controles com as restrições de integridade do modelo relacional.

<table>
<thead>
<tr>
<th>Relacional</th>
<th>XSD</th>
<th>Conteúdo de elementos</th>
<th>Sequência e escolha de elementos</th>
<th>Padrões de elementos</th>
<th>Ocorrência de elementos</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) Domínio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tipo</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Qualquer conteúdo, Elementos vazios, Conteúdo simples, Conteúdo misto, Elementos referência, Elementos complexos, Elementos ANY, Elementos GROUP</td>
<td>Elemento SEQUENCE, Elemento CHOICE, Modelo complexo (SEQUENCE e CHOICE)</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td>Elementos opcionais repetitivos (maxOccurs e minOccurs), Elemento ALL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Atributos</td>
<td>–</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tuplas</td>
<td>–</td>
<td>–</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Tabela 4.2- Comparativo XSD vs relacional (I). |
Tabela 4.2 - Comparativo XSD vs relacional (II).

<table>
<thead>
<tr>
<th></th>
<th>Relacional</th>
<th>XSD</th>
<th>Conteúdo de atributos</th>
<th>Padrões de atributos</th>
<th>Ocorrência de atributos</th>
<th>Tipos de dados</th>
<th>Chaves e referências</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>a) Domínio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tipo</td>
<td></td>
<td>Em elementos vazios,</td>
<td>–</td>
<td>–</td>
<td>Tipos simples</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Em elementos com</td>
<td></td>
<td>–</td>
<td>(SIMPLETYPE,</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>conteúdo,</td>
<td></td>
<td>–</td>
<td>FACETS), Tipos</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Pertencentes a</td>
<td></td>
<td>–</td>
<td>complexos</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>namespaces</td>
<td></td>
<td>–</td>
<td>(Devivações por</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Atributos</td>
<td></td>
<td></td>
<td>Com valores padrão</td>
<td>Requeridos e</td>
<td>extensão e</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(DEFAULT), Com</td>
<td>opcionais (USE)</td>
<td>restrição)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>valores fixos</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(FIXED), Valores</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>enumerados</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(ENUMERATION)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Tuplas</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Banco de dados</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td></td>
</tr>
<tr>
<td></td>
<td>b) Chaves</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Candidatas</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>UNIQUE, KEY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Primárias</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>KEY</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Estrangeiras</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>KEYREF</td>
</tr>
<tr>
<td></td>
<td>c) Integridade referencial</td>
<td></td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>UNIQUE, KEY, KEYREF</td>
</tr>
<tr>
<td></td>
<td>d) Momento de Verificação</td>
<td></td>
<td>IMEDIATO</td>
<td>IMEDIATO</td>
<td>IMEDIATO</td>
<td>IMEDIATO</td>
<td></td>
</tr>
<tr>
<td></td>
<td>e) Baseada em eventos</td>
<td></td>
<td>SAX/ DOM</td>
<td>SAX/ DOM</td>
<td>SAX/ DOM</td>
<td>SAX/ DOM</td>
<td></td>
</tr>
</tbody>
</table>

Analisando de uma maneira geral as Tabelas 4.2 (I) e 4.2 (II), nota-se que a maioria dos controles do XSD se enquadra no tipo de restrição de domínio do modelo...
relacional, tendo principalmente controles para tratar restrições de tipo e restrições de atributos.

Os controles de conteúdo de elementos, controles de seqüência e escolha de elementos, controles de ocorrência 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 eles estão associados.

A especificação de tipos de dados também pode ser associada ao modelo relacional como uma restrição de tipo, porque 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, nas Tabelas 4.2 (I) e 4.2 (II), 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.

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 e 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, ao contrário do elemento UNIQUE, cujo seu 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. 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 e um *parser* que suporte essa linguagem, porém estas não fazem parte da especificação de esquemas, e, sim, da tecnologia XML.

Através das Tabelas 4.2 (I) e 4.2 (II) também se constata que, na especificação do XSD, não existem especificações de controles de restrições de tuplas e restrições de banco de dados semelhantes às existentes no modelo relacional. Elas permitem definir restrições de integridade de domínio mais complexas.

### 4.2 Trabalhos Relacionados

Como demonstrado na seção 4.1, a especificação de esquemas XML não é expressiva o suficiente para suportar todos os tipos de restrições de integridade de bancos de dados, mais especificamente, do modelo relacional. Em virtude disso, 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, pode-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 sempre do tipo IMEDIATO. A seguir são citados alguns trabalhos referentes a esses grupos de restrições, com detalhamento nos trabalhos referentes a restrições de integridade de domínio e restrições de integridade baseada em eventos. O foco desta dissertação são as restrições de integridade de domínio, porém utiliza-se características das restrições de integridade baseadas em eventos para tratá-las.

No que se refere a restrições de domínio, geralmente são impostas através de linguagens de definição de esquemas. Em DONGWON & CHU (2000) é feito um estudo comparativo entre as seguintes linguagens: DTD, XSD, XDR, SOX, Schematron e DSD. Os critérios de classificação nos quais é baseado o estudo comparativo são: (i) Schema, onde são avaliados os itens: sintaxe XML, suporte a *namespaces*, importação e exclusão de fragmentos externos; (ii) tipos de dados, que são divididos em tipos simples e tipos complexos, estes se subdividindo em: construção de tipos, definição de tipos
pelo usuário, restrição no tipo de dados e diferenciação de valores nulos; (iii) atributos, em que são avaliados os itens: valor default, escolha entre atributos, opcional vs. obrigatório, restrição dos valores, definição de condições; (iv) elementos, onde são avaliados os itens: valor default, definição do modelo do conteúdo, permitir sequências ordenadas e desordenadas, escolha, ocorrências mínimas e máximas, modelo aberto, definições condicionais; (v) herança, que é subdividida em: tipos complexos por extensão e restrição e tipos simples por restrição e extensão; (vi) unicidade ou chave, em que são avaliados os seguintes itens: unicidade de atributos e não-atributos, chave para atributos e não-atributos, chave estrangeira para atributos e não atributos; (vii) características gerais. Nesse último caso são avaliados: restrição dinâmica (pode-se, seletivamente, ativar e desativar as restrições em partes do esquema, as quais serão avaliadas dinamicamente), permitir diferentes definições de elementos e atributos com o mesmo nome, documentação, permitir fragmentos html e ser autodescritiva.

![Diagrama de Classificação de Linguagens de Esquemas XML](image)

**Classe 1**
- **DTD**: Tem o suporte básico para estrutura

**Classe 2**
- **SOX**: Incorpora XDR e tem: herança, mais tipos de dados e modularização
- **XDR**: Incorpora DTD e tem: mais estrutura, mais modelos contidos e mais tipos de dados

**Classe 3**
- **XSD**: Incorpora SOX e tem: mais estrutura, tipos de dados, herança, unicidade e chaves
- **DSD**: Incorpora SOX e tem: restrições, mais estrutura, documentação e versão
- **Schematron**: Incorpora SOX e tem: restrição, unicidade e chaves, documentação


Figura 4.1- Classificação de linguagens de esquemas XML.
Ao final, é feita uma classificação em que são definidas três classes (Classe 1, Classe 2 e Classe 3), baseada na quantidade de recursos que linguagens estudadas disponibilizam. A Classe 1 indica a menor quantidade de recursos, sua estrutura de esquema é mínima e suporta poucos tipos de dados e restrições. A Classe 2 possui uma característica intermediária, suporta mais tipos de dados que a Classe 1, porém ainda é insuficiente, pois possui poucos controles de imposição de restrições. A Classe 3 possui a maior quantidade de recursos entre as classes: suporta tipos de dados e estruturas de esquemas mais complexas. A Figura 4.1 mostra a respectiva classificação.

OGBUJI (2001) apresenta uma linguagem de validação para documentos XML chamada Schematron (Classe 3 da Figura 4.1). Esta linguagem, ao invés de declarar esquemas, define uma série de regras de validação que são aplicadas dentro de um XSD e o validam contra uma instância do documento XML, como mostra a Figura 4.2.

```xml
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
	xmllns:xmlns:xmls="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" 
            diagnostcis="lessThan">
            A should be greater than B.
          </sch:assert>
        </sch:rule>
      </sch:pattern>
      <sch:diagnostics id="lessThan">
        Error! A is less than B
        A = <sch:value-of select="d:A"/>
        B = <sch:value-of select="d:B"/>
      </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>
```


Figura 4.2- Exemplo de esquema XSD na linguagem Schematron.
Schematron depende quase que exclusivamente de expressões XPath para definir as 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 estas validações no documento XML.

A Figura 4.2 mostra um exemplo de esquema XSD utilizando a linguagem Schematron. O objetivo desse documento 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 extrairá as informações de saída do documento XSD e criará um esquema Schematron que será validado contra a instância do documento.

Outra abordagem, proposta por PROVOST (2002), para a imposição de restrições de domínio em documentos XML é através da combinação de linguagens de estilo XSLT (XML Stylesheet Language Transformations) com XPath. PROVOST (2002) cita que, para impor restrições, são necessários 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 4.3 mostra um exemplo de documento XSLT, que tem como objetivo verificar se o valor do elemento A é menor que o valor do elemento B.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
               xmlns:d="http://www.demo.org"
               version="1.0">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:if test="/d:Demo/d:A < /d:Demo/d:B">
      <xsl:text>Error! A is less than B</xsl:text>
      <xsl:if test="/d:Demo/d:A = /d:Demo/d:B">
        <xsl:text>A = </xsl:text>
      </xsl:if>
    </xsl:if>
    <xsl:if test="/d:Demo/d:A >= /d:Demo/d:B">
      <xsl:text>Instance document is valid</xsl:text>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

Fonte: PROVOST (2002).

Figura 4.3- Exemplo de restrições usando XSLT e XPath.
BENEDIKT et al. (2002) descreve 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 4.4 mostra um exemplo de definição de restrição na linguagem ΔX, que, quando anexado ao elemento regiao no esquema XSD, expressa que a capacidade máxima de uma região deve ser a mesma capacidade das regiões relacionadas.

```
<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 4.4- Exemplo da linguagem ΔX.

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.

BAYLEY et al. (2002) mostra uma abordagem de validação de restrições de integridade através de regras ECA (Event-Condition-Action), 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 4.5 (b) mostra uma regra ECA que irá
atualizar o elemento livros_em_estoque no documento da Figura 4.5 (a) após um ou mais livros serem inseridos no documento.

Nos exemplos da Figura 4.5 (b) e Figura 4.5 (c), a variável do sistema $\delta$ contém uma série de novos nós 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. Se é, uma cópia do elemento título é inserida como filho do elemento livros_em_estoque. A Figura 4.5 (c), similaremnte, atualiza o elemento livros_em_estoque após a exclusão de um ou mais livros no documento.

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

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

(b)
on DELETE /livraria/livro
if TRUE
do DELETE titulo[.=delta/titulo] BELOW /livraria/livros_em_estoque

(c)
```

Fonte: Adaptada de BAYLEY et al. (2002). Figura 4.5- Exemplos de regras ECA.
Existem trabalhos que tratam as questões de 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 e chaves estrangeiras no mapeamento de documentos XML para bancos relacionais sobre a melhor maneira de efetuar a validação delas: se através do modelo relacional ou através de verificadores de restrições XML nativos. Além disso, BUNEMAN et al. (2002) trabalha com o conceito de chaves relativas\textsuperscript{11} no contexto de bancos de dados científicos e estruturas hierárquicas. Porém, esses trabalhos não fazem parte do escopo desta dissertação.

A Tabela 4.3 compara os trabalhos relacionados com relação às restrições de integridade de domínio e restrições de integridade baseadas em eventos do modelo relacional. A Tabela 4.4 compara os trabalhos relacionados com os recursos de imposição de restrições de integridade de domínio e restrições de integridade baseadas em eventos da SQL. SQL oferece suporte, através de gatilhos (TRIGGERS), assertivas (ASSERTIONS) e a cláusula CHECK, para a especificação de restrições de integridade de domínio em bancos de dados relacionais e pode ser usada como base para a especificação de restrições de integridade de domínio em outros modelos de dados. Essas comparações mostram alguns pontos importantes a serem salientados.

\textbf{Tabela 4.3- Comparativo trabalhos vs. relacional.}

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Relacional</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Domínio</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tipo</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td>Atributos</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Tuplas</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Banco de dados</td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td>Baseada em eventos</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
</tr>
</tbody>
</table>

\textsuperscript{11} Uma chave relativa consiste em um par \((Q, K)\), onde \(Q\) é uma expressão de caminho e \(K\) é a chave.
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 que são validados por um parser específico (OGBUJI, 2001).

Os padrões de consulta XPath são utilizados nas linguagens de validação e podem ser associados a uma linguagem de estilo XSLT em busca da validação de documentos XML (PROVOST, 2002).

BAYLEY et al. (2002) define uma linguagem baseada em regras ECA para inserir e apagar elementos em documentos XML. Possui sintaxe semelhante a um gatilho (trigger), onde busca incorporar restrições de integridade baseadas em eventos em documentos XML. Porém, não é utilizada para tratar restrições de integridade de domínio.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Domain</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Check</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Assertion</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td><strong>Gatilhos</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>√</td>
</tr>
<tr>
<td><strong>Procedimentos</strong></td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>-</td>
</tr>
<tr>
<td><strong>Transações</strong></td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Analisando a Tabela 4.4, nota-se que 75% dos trabalhos relacionados possuem recursos semelhantes aos procedimentos existentes na linguagem SQL, ou seja, são instruções escritas que são executadas através de uma chamada de aplicação. Apenas BAYLEY et al. (2002) trabalha com um recurso semelhante a gatilhos, onde, em conseqüência da ocorrência de um evento e uma determinada condição, são inseridos ou apagados elementos num documento XML. Nenhum dos trabalhos relacionados possui
especificações semelhantes a *domain, checks, assertion* e transações, existentes na linguagem SQL.

Os trabalhos relacionados, com relação a restrições de integridade de domínio do modelo relacional (Tabela 4.3) e com relação aos recursos de imposição de restrições de integridade de domínio da SQL (Tabela 4.4), demonstram algumas limitações. Alguns pontos importantes devem ser salientados:

- apenas um trabalho relacionado, BAYLEY *et al.* (2002), possui um enfoque equivalente à maneira que a linguagem SQL trata restrições de integridade de domínio, porém serve apenas para inserir e apagar elementos. Nenhum dos outros trabalhos relacionados levam 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 é referente ao entendimento do esquema, pois ele contém elementos adicionais para a imposição de restrições;
- pode-se validar restrições de integridade de domínio em documentos XML através do uso conjunto de XSLT e *XPath* (PROVOST, 2002). Porém, essa especificação tende ser extensa, pois define um processamento recursivo sobre o documento XML.

Essas limitações encontradas nos trabalhos relacionados motivaram o desenvolvimento desta dissertação, cuja proposta, apresentada no próximo capítulo, visa atender aos requisitos recém-mentionados.
5 Incorporação de Restrições de Integridade de Domínio em Documentos XML

Baseado na motivação estabelecida no capítulo 4, este capítulo propõe uma linguagem chamada XDCL (XML Domain Constraint Language) para a incorporação de restrições de integridade de domínio em documentos XML. A seguir são apresentados a arquitetura de validação de documentos XML sobre a qual XDCL se aplica, um estudo de caso e as atividades futuras.

5.1 Arquitetura da Abordagem

O principal objetivo desta dissertação é propor uma extensão para a tecnologia XML, visando o tratamento de restrições de integridade de domínio em documentos XML, em específico, restrições de tuplas e restrições de banco de dados. Para permitir a validação de restrições de um documento XML, além das especificações do W3C, é necessária uma linguagem que defina restrições e uma ferramenta que valide essas restrições para um determinado documento XML.

A arquitetura proposta pode funcionar como um recurso adicional de validação de restrições de domínio para aplicações que utilizam a tecnologia XML. Ela pode ser adotada tanto numa aplicação quanto num banco de dados XML nativo.

A Figura 5.1 mostra a arquitetura da abordagem proposta e a maneira pela qual ela se insere no contexto de um ambiente de gerenciamento de dados XML. Nela, a linguagem para a incorporação de restrições de integridade chama-se XDCL (XML Domain Constraint Language) e a ferramenta de validação dessas restrições é chamada de parser XDCL.
Figura 5.1 - Arquitetura da abordagem proposta.

Na linguagem XDCL serão expressas as restrições de integridade de domínio levantadas a partir dos requisitos da análise do domínio da aplicação, as quais serão armazenadas num documento textual que terá a extensão XDC. Um documento XDC segue a sintaxe XML. A ideia é que a sintaxe utilize a maior quantidade possível de
recursos já existentes na tecnologia XML para a definição de restrições de domínio, como expressões XPath e XSLT.

Na arquitetura proposta, inicialmente um parser XML irá validar um documento XDC definido com base nas restrições de domínio da aplicação, verificando se ele é **válido e bem-formado**. Um documento XML ou um banco de dados XML nativo que deseja estar de acordo com as restrições impostas pelo documento XDC fará referência a ele.

Um documento XDC irá referenciar um esquema XML (DTD ou XSD), visto que os elementos e atributos utilizados nas expressões da linguagem XDCL deverão, necessariamente, pertencer a um esquema XML. Assim sendo, não será necessário expressar na linguagem XDCL restrições de integridade de tipo e restrições de integridade de atributo, pois o esquema XML referenciado já as impõe. A linguagem XDCL procurará expressar restrições de integridade classificadas como restrições de tuplas e restrições de banco de dados.

Para ilustrar as restrições de integridade que a linguagem XDCL será capaz de impor, a Figura 5.2 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.

Um exemplo de restrição que deverá ser considerada e validada neste documento é que o conteúdo do elemento **data_lancamento** de uma consulta não pode ser uma data posterior à contida no elemento **data_realizacao** da mesma consulta. Este caso caracteriza uma restrição de integridade de tupla.

Outro tipo de restrição a ser considerada e validada 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 utilizará como base recursos semelhantes aos existentes na linguagem SQL, principalmente através do uso
de cláusulas semelhantes a gatilhos (triggers), asserções (assertion) e cláusulas de checagem (checks), inexistentes na tecnologia XML.

```xml
<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>29-04-2004</data_geracao>
    <total_pagamentos>54,00</total_pagamentos>
    <consultas>
      <qtde_consultas>2</qtde_consultas>
      <total_consultas>54,00</total_consultas>
      <consulta>
        <autorizacao>813321</autorizacao>
        <data_lancamento>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>12-04-2004</data_realizacao>
        <quantidade>1</quantidade>
        <valor_consulta>27,00</valor_consulta>
      </consulta>
      <consulta>
        <autorizacao>81341</autorizacao>
        <data_lancamento>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>20-04-2004</data_realizacao>
        <quantidade>1</quantidade>
        <valor_consulta>27,00</valor_consulta>
      </consulta>
    </consultas>
  </prestadores>
</dados>
```

Fonte: Primária.

Figura 5.2- Exemplo de documento XML onde restrições de integridade podem ser definidas.

A ferramenta para validar as restrições XDCL num documento XDC será o parser XDCL. Ele irá validar as expressões XDCL contra as instâncias XML existentes num documento XML ou por um banco de dados nativo XML, que referenciam o
documento XDC. Após ser feita a validação, o *parser XDCL* mostrará o resultado da validação, que poderá ser uma mensagem de validação correta ou indicações de inconsistências existentes nas instâncias.

### 5.2 Estudo de caso

A proposta será validada no domínio de uma operadora de planos de saúde dos hospitais Prontoclínicas Ltda e SP Ltda, com sede, respectivamente, nas cidades de Passo Fundo-RS e Lagoa Vermelha-RS. Uma operadora de plano de saúde tem como objetivo prestar atendimento à saúde de seus usuários, como consultas, exames, internamentos e procedimentos médicos. Quem realiza esses atendimentos são 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. Pretende-se realizar essa troca de dados através de documentos XML, que serão validados por um esquema XSD e de uma especificação XDCL. Um exemplo de documento XML para este domínio é o mostrado na Figura 5.2. Com essas validações pretende-se minimizar o tempo desperdiçado com erros existentes nos arquivos recebidos pela operadora de plano de saúde, garantindo uma maior consistência e correção dos dados recebidos.

Esses arquivos poderão ou não ser importados para o banco de dados relacional existente na operadora. No caso de serem importados, já estarão consistentes, minimizando a carga de validação semântica durante o processo de armazenamento. Caso contrário, a partir desses documentos poderão ser geradas novas instâncias XML, que serão processadas por outros aplicativos. Mesmo assim, é necessário também que eles estejam consistentes.

### 5.3 Atividades futuras

A Tabela 5.1 apresenta um cronograma das atividades a serem ainda realizadas para a conclusão desta dissertação.
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Defesa TI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adequação da proposta com sugestões da Banca do TI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Estudo detalhado da sintaxe de checks, triggers e assertions da SQL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Definição da linguagem XDCL com base no estudo da etapa anterior</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementação do parser XDCL</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Validação</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Redação da dissertação</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Defesa da dissertação</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Referências


2. 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. 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.

5. BENEDIKT, Michael; CHAN, Chee-Young; FAN, Wenfei; FREIRE, Juliana; RATOGI, Rajeev. Capturing types and constraints in Data Exchange. Em SIGMOD 2003.


## Anexos

### Anexo 1  Tipos Simples x Facets

<table>
<thead>
<tr>
<th>Tipos simples</th>
<th>Facets</th>
<th>length</th>
<th>minLength</th>
<th>maxLength</th>
<th>pattern</th>
<th>enumeration</th>
<th>whiteSpace</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>normalizedString</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>token</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>byte</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>unsignedByte</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>base64Binary</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>hexBinary</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>integer</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>positiveInteger</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>negativeInteger</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>nonNegativeInteger</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>nonPositiveInteger</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>int</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>unsignedInt</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>long</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>unsignedLong</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>short</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>unsignedShort</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>decimal</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>float</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>double</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>boolean</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>time</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>dateTime</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>duration</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>date</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>gMonth</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>gYear</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>gYearMonth</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>gDay</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>gMonthDay</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>Name</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>QName</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>NCName</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>anyURI</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>language</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>ID</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>IDREF</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>IDREFS</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>ENTITY</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>ENTITIES</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>NOTATION</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>NM_TOKEN</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>NM_TOKENS</td>
<td></td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
<td>y</td>
</tr>
</tbody>
</table>
Anexo 2 Tipos Simples x Facets enumeradas.

<table>
<thead>
<tr>
<th>Tipos Simples</th>
<th>Facets</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>max Inclusive</td>
</tr>
<tr>
<td>byte</td>
<td>y</td>
</tr>
<tr>
<td>unsignedByte</td>
<td>y</td>
</tr>
<tr>
<td>integer</td>
<td>y</td>
</tr>
<tr>
<td>positiveInteger</td>
<td>y</td>
</tr>
<tr>
<td>negativeInteger</td>
<td>y</td>
</tr>
<tr>
<td>nonNegativeInteger</td>
<td>y</td>
</tr>
<tr>
<td>nonPositiveInteger</td>
<td>y</td>
</tr>
<tr>
<td>int</td>
<td>y</td>
</tr>
<tr>
<td>unsignedInt</td>
<td>y</td>
</tr>
<tr>
<td>long</td>
<td>y</td>
</tr>
<tr>
<td>unsignedLong</td>
<td>y</td>
</tr>
<tr>
<td>short</td>
<td>y</td>
</tr>
<tr>
<td>unsignedShort</td>
<td>y</td>
</tr>
<tr>
<td>decimal</td>
<td>y</td>
</tr>
<tr>
<td>float</td>
<td>y</td>
</tr>
<tr>
<td>double</td>
<td>y</td>
</tr>
<tr>
<td>time</td>
<td>y</td>
</tr>
<tr>
<td>dateTime</td>
<td>y</td>
</tr>
<tr>
<td>duration</td>
<td>y</td>
</tr>
<tr>
<td>date</td>
<td>y</td>
</tr>
<tr>
<td>gMonth</td>
<td>y</td>
</tr>
<tr>
<td>gYear</td>
<td>y</td>
</tr>
<tr>
<td>gYearMonth</td>
<td>y</td>
</tr>
<tr>
<td>gDay</td>
<td>y</td>
</tr>
<tr>
<td>gMonthDay</td>
<td>y</td>
</tr>
</tbody>
</table>