

UNIVERSIDADE PRESBITERIANA MACKENZIE

**FLÁVIO JIMENEZ VÉRDI DE FIGUEIREDO
HILTON HIGASI
LINCOLN MASAO MURAOKA JUNIOR**

XML E BANCOS DE DADOS

São Paulo
2003

SUMÁRIO

<u>1. INTRODUÇÃO</u>	4
<u>2. XML</u>	8
2.1 ORIGEM DA XML	8
2.2 CONCEITOS BÁSICOS	9
2.3 DEFINIÇÃO DA ESTRUTURA DE UM DOCUMENTO XML	14
2.3.1 Document Type Definition (DTD)	14
2.3.2 XML Schemas Definition	15
2.4 TECNOLOGIAS RELACIONADAS COM XML	17
2.4.1 SAX	17
2.4.2 DOM	18
2.4.3 XPath	18
2.4.4 XSL	18
<u>3. BANCOS DE DADOS E XML</u>	19
3.1 DEFINIÇÃO DE BANCOS DE DADOS	19
3.2 HISTÓRIA E EVOLUÇÃO DOS BANCOS DE DADOS	21
3.3 TIPOS DE BANCOS DE DADOS	23
3.3.1 Bancos de dados hierárquicos	23
3.3.2 Bancos de dados de redes	23
3.3.3 Bancos de dados relacionais	24
3.3.4 Bancos de dados orientados a objeto	24
3.3.5 Bancos de dados objeto-relacionais	24
3.3.6 Bancos de dados nativos em XML	25
3.4 XML, O NOVO PARADIGMA	26
3.5 CARACTERÍSTICAS DA XML PARA BANCOS DE DADOS	29
3.5.1 Desenvolvimento de padrões de uso específico	29
3.5.2 Padrão para visões de bancos de dados	30
3.5.3 Separação entre estrutura, conteúdo e materialização dos dados	31
<u>4. ARMAZENAMENTO DE DADOS XML</u>	32
4.1 A XML NO CONTEXTO ATUAL DOS BANCOS DE DADOS	32
4.2 GRAU DE ESTRUTURAÇÃO	33
4.2.1 Informações voltadas para dados	33
4.2.2 Informações voltadas para documentos	34
4.2.3 Mas porque usar dados semi-estruturados?	35
4.2.4 Dados versus documentos em bancos de dados	36
4.3 SOLUÇÕES PARA INFORMAÇÕES VOLTADAS A DADOS	37
4.3.1 Middleware de conversão	38
4.3.2 Extensões em bancos de dados objeto-relacionais	38
4.4 SOLUÇÕES PARA INFORMAÇÕES VOLTADAS A DOCUMENTOS	39
4.4.1 Arquivo XML como repositório de dados	40
4.4.2 Uso do sistema de arquivos	41
4.4.3 Sistemas relacionais para armazenamento de documentos XML	42
4.4.4 Sistemas de bancos de dados nativos em XML	43
<u>5. XML EM SISTEMAS DE BANCOS DE DADOS COMERCIAIS</u>	46
5.1 MIDDLEWARE	46

<u>5.1.1 DB2XML</u>	46
<u>5.1.2 BeanStalk</u>	46
<u>5.1.3 Allora</u>	47
<u>5.2 SISTEMAS RELACIONAIS COM SUPORTE PARA XML</u>	47
<u>5.2.1 DB2</u>	47
<u>5.2.2 Oracle 9i</u>	48
<u>5.2.3 SQL Server 2000</u>	49
<u>5.3 SISTEMAS NATIVOS EM XML</u>	50
<u>5.3.1 Tamino XML Server</u>	50
<u>5.3.2 Lore</u>	50
<u>6. INTRODUÇÃO AO XINDICE</u>	52
<u>6.1 INTRODUÇÃO</u>	52
<u>6.2 CARACTERÍSTICAS</u>	52
<u>6.3 COMO OBTER UMA DISTRIBUIÇÃO</u>	54
<u>6.4 REQUISITOS PARA INSTALAÇÃO</u>	54
<u>6.5 INSTALANDO O XINDICE</u>	54
<u>6.6 COMANDOS DE OPERAÇÕES</u>	55
<u>6.6.1 Criando uma coleção</u>	55
<u>6.6.2 Adicionando um documento a uma coleção</u>	56
<u>6.6.3 Recuperando dados</u>	57
<u>6.6.4 Atualizando registros</u>	57
<u>6.6.5 Criando índices</u>	59
<u>6.7 CONSIDERAÇÕES</u>	60
<u>7. IMPLEMENTAÇÃO</u>	61
<u>7.1 APRESENTAÇÃO / MODELAGEM DO SISTEMA</u>	61
<u>7.2 PLATAFORMA</u>	64
<u>7.3 ARQUITETURA DO SISTEMA</u>	64
<u>7.4 O SISTEMA EM AÇÃO</u>	70
<u>7.5 RESULTADOS</u>	78
<u>8. CONCLUSÃO</u>	79
<u>REFERÊNCIAS BIBLIOGRÁFICAS</u>	82
<u>BIBLIOGRAFIA COMPLEMENTAR</u>	84
<u>APÊNDICE A – CÓDIGOS FONTE DA IMPLEMENTAÇÃO</u>	85

1. INTRODUÇÃO

A XML (eXtensible Markup Language, ou Linguagem Extensível de Marcação) foi criada em 1996, por especialistas do World Wide Web Consortium (W3C), órgão de regulamentação dos padrões utilizados na internet. Ela fornece um formato estruturado para descrição de dados e foi criada como um subconjunto simplificado do padrão SGML (Standard Generalized Markup Language, ou Linguagem Padrão de Marcação Generalizada).

Características de flexibilidade e portabilidade vêm fazendo com que, nos últimos anos, a XML seja aceita como um padrão para representação, intercâmbio e manipulação de dados em aplicações para as mais diversas áreas de negócios. Representação de dados em aplicações de gerenciamento de conteúdo, aplicações de transações bancárias e de publicação de conteúdo em intranets são alguns exemplos de uso da XML.

O crescimento na utilização da XML levou a um aumento significativo no volume de dados nesse formato que são armazenados, transportados e recuperados por aplicações no mundo todo. Um exemplo que ilustra essa nova demanda é a troca de dados entre dois sistemas de bancos de dados diferentes, usando documentos XML para o intercâmbio dos dados.

Essa nova realidade faz surgir novos questionamentos a respeito de onde e como armazenar esses dados, e qual a melhor maneira de integrar as tecnologias já existentes de sistemas de bancos de dados (principalmente no modelo relacional) com a XML. Também surgiram novas tecnologias para armazenamento de XML a serem consideradas, como os bancos de dados nativos de XML.

Esse trabalho de pesquisa apresenta um panorama das formas mais adotadas atualmente para o armazenamento de dados no formato XML. Diversas abordagens têm sido propostas para este novo desafio. Serão conceituadas e analisadas características de cada abordagem, destacando vantagens e desvantagens. Como exemplo de uso de banco de dados

XML nativo, apresentaremos a implementação de uma pequena aplicação fazendo uso desta nova tecnologia.

O presente trabalho está dividido em oito capítulos e um apêndice, abordando os conteúdos descritos a seguir.

O capítulo 1 introduz o tema do trabalho e apresenta a questão a ser abordada, justificando-a e limitando-a.

O capítulo 2 traz um breve histórico da XML, a definição do que é a XML, e expõe alguns conceitos básicos sobre ela.

O capítulo 3 apresenta conceitos de bancos de dados e um breve histórico dos modelos de bancos de dados, para em seguida fazer a relação entre bancos de dados e XML. São apresentadas características da XML relacionadas à área de bancos de dados.

O capítulo 4 mostra os diferentes tipos de estruturação de documentos em XML, descrevendo-os e comparando-os entre si, para em seguida tratar das soluções de armazenamento de XML de acordo com a forma de estruturação dos documentos.

No capítulo 5 são apresentadas algumas implementações de sistemas de bancos de dados comerciais que lidam com dados no formato XML.

O capítulo 6 apresenta o Xindice, um sistema de bancos de dados nativo em XML, descrevendo suas principais características e funcionalidades.

Já o capítulo 7 mostra a implementação de um banco de dados simples fazendo uso do Xindice.

O capítulo 8, por fim, traz as conclusões tiradas da implementação do capítulo anterior e da análise do tema como um todo.

O Apêndice A compreende alguns códigos fonte para ilustrar como foi feita a implementação do sistema utilizando a linguagem Java junto com o banco de dados Xindice.

2. XML

A XML é das tecnologias recentes mais promissoras, seja simplesmente para marcação, para intercâmbio, ou para estruturação e armazenamento de dados. Este capítulo apresenta conceitos básicos sobre esta linguagem, e define algumas tecnologias que são usadas em conjunto com a XML, estendendo-a e complementando-a.

2.1 ORIGEM DA XML

A eXtensible Markup Language (XML), assim como a HyperText Markup Language (HTML), tem sua origem na SGML. O SGML é um padrão internacional, declarado na especificação ISO8879, de marcação para documentos. Fornece um esquema de marcação simples, independente de plataforma e extremamente flexível. As três linguagens têm em comum o ML (Markup Language) na sigla. Elas são linguagens de marcação, ou seja, são utilizadas para marcar um determinado documento, atribuindo tanto especificações de como o texto delimitado deve ser exibido como acrescentando informações explicativas sobre o texto delimitado.

Marcação é qualquer informação adicional acrescentada ao texto de um documento. E não é rara a sua utilização. Por exemplo, todos os processadores de texto inserem marcadores nos documentos criados. Estes marcadores são usados para controlar a aparência do documento e representam itens como tipo e tamanho da fonte, negrito, sublinhado, itálico etc. Ao contrário da marcação comum, utilizada em processadores de texto, que existe para o controle da aparência de um documento, a marcação generalizada tem uma filosofia muito mais simples: indicar a importância estrutural de uma porção de texto dentro de um documento.

A SGML é uma linguagem de marcação bastante robusta, porém a complexidade

inerente à qualquer recurso da SGML limita a base de seus usuários, tornando sua utilização bastante restrita a grandes empresas e para aplicações específicas, como aplicações em medicina ou matemática. A HTML, na verdade, é uma aplicação da SGML para a marcação de conteúdo, de modo a prepará-lo para apresentação na World Wide Web.

Em 1996, o W3C iniciou um projeto de desenvolvimento de uma linguagem de marcação abrangente, que reunisse a flexibilidade do SGML com a ampla aceitação do HTML, para uso através da internet. Surgiu então a XML, um subconjunto da SGML, extensível como tal, porém de forma muito mais simplificada.

2.2 CONCEITOS BÁSICOS

A XML é uma linguagem baseada em elementos de marcação de sintaxe, ou simplesmente tags, conceitualmente muito próximos das tags da HTML. Entretanto, a HTML possui um conjunto pré-definido e fixo de tags, enquanto a XML oferece a possibilidade de definir seus próprios elementos e a funcionalidade dos mesmos. Ela baseia seu formato em um arquivo de texto, com estruturas organizadas hierarquicamente.

Suponhamos que temos que usar uma linguagem de marcação para representar uma carta. Em HTML poderíamos ter algo como:

```
<p>
  <font face="verdana" size=10>
    <b>Remetente:</b> José da Silva<br>
    <b>Destinatária:</b> Maria daSilva<br><br>
  </font>
</p>
<p>
  <font face="arial" size=12>
    Olá, Maria, bom dia!<br>
  </font>
</p>
<p>
  <font face="arial" size=12>
    Espero notícias suas! Até mais.<br>
  </font>
</p>
```

Exemplo 1: Trecho de documento em HTML

Olhando para esse exemplo, pode-se reconhecer facilmente que se trata de uma carta. Só chegamos a essa conclusão, entretanto, porque relacionamos a estrutura deste documento com nosso conhecimento prévio do que entendemos por "carta". Já uma aplicação de software (ou mesmo alguém que não tenha conhecimento desta estrutura para redação de cartas) não teria esta mesma condição. O motivo disto é que as únicas informações disponíveis a respeito da estrutura deste documento, além dos dados em si, são as "tags" da HTML: a tag <p> diz ao aplicativo interpretador de HTML, comumente um navegador de internet, que ele deve efetuar uma quebra de linha e inserir uma linha em branco antes de continuar a apresentação dos dados. A tag
 significa ao aplicativo que deve ser inserida uma quebra de linha e retorno de carro. Já a tag marca aquilo que será mostrado em negrito, e a tag controla características da fonte a ser utilizada para apresentação. Em resumo, as tags dizem respeito à apresentação dos dados, ou seja, dão conta apenas do modo como os dados serão exibidos, por exemplo, no navegador de internet.. Em momento algum elas fornecem informação a respeito do conteúdo dos dados sendo apresentados.

Não seria prático nem eficaz desenvolver um algoritmo que associasse a posição das tags HTML do exemplo ao significado semântico dos dados marcados por elas. Em outros termos: elaborar um algoritmo qualquer para determinar que a primeira ocorrência da tag <p> diz respeito ao cabeçalho da carta, por exemplo, não seria funcional. Um analisador de estrutura usando tal algoritmo limitaria o uso da tag <p> para criação de cabeçalhos da carta, não podendo ser mais usada tal tag no restante do documento. Como o conjunto de tags da HTML é limitado, tal abordagem "engessada" acabaria por limitar quaisquer ampliações que se fizessem necessárias nesta estrutura.

Esta limitação da HTML foi endereçada pela XML, e é tão vital que é mencionada em seu nome: o "X" de XML significa "eXtensible", ou extensível. Significa que pode-se desenvolver para cada aplicação da XML um subconjunto próprio de tags. Essas tags

receberiam nomes que nada dizem a respeito da apresentação dos dados, apenas representam uma estrutura determinada no contexto deste documento. A mesma carta do exemplo 1 poderia ser representada, utilizando XML, da seguinte maneira:

```
<carta tipo="social">
  <cabeçalho>
    <remetente> José da Silva </remetente>
    <destinatário> Maria da Silva </destinatário>
  </cabeçalho>
  <parágrafo>
    Olá, Maria, bom dia!
  </parágrafo>
  <parágrafo>
    Espero notícias suas! Até mais.
  </parágrafo>
</carta>
```

Exemplo 2: Trecho de documento em XML

No exemplo 2, temos uma marcação que atende à estruturação de um tipo de documento em específico, uma carta nesse exemplo. Nada nesse documento mostra informações a respeito de como os dados serão apresentados, mas temos uma visão muito mais clara a respeito de sua estrutura.

Utilizando o exemplo 2, passaremos por algumas definições de termos da linguagem XML, que serão utilizados no decorrer deste texto. Conforme mencionado, a linguagem XML permite a criação de documentos, e para tanto, faz uso principalmente de elementos e atributos, além dos dados em si.

Um arquivo XML em sua forma mais simples é um arquivo de texto contendo algum conteúdo e marcação que especifica aquele conteúdo. Pode estar presente alguma informação adicional sobre a estrutura deste arquivo. É mais usado o termo “documento XML” do que “arquivo XML” para descrever algum conteúdo com marcação XML. Um documento XML sempre tem um elemento para delimitá-lo. No nosso exemplo, o elemento <carta> determina o início e fim de um documento. Assim, dentro de um mesmo arquivo XML, podem existir

vários documentos.

Os elementos são representados por tags de início e fim. O nome de um elemento é o texto que está entre os sinais de maior e menor. Diferente da HTML, um elemento deve necessariamente ter tags de início e final. O dado que este elemento representa aparece entre as tags de início e fim. Um exemplo ilustrativo pode ser dado por: <parágrafo> Olá, Maria, bom dia! </parágrafo>.

Um atributo é uma informação descritiva do elemento, e consiste de um nome e de um valor. Os atributos devem ser especificados sempre entre delimitadores, que podem ser aspas duplas ou simples. Em <carta tipo="simples">, o elemento <carta> possui um atributo chamado tipo, cujo valor é "simples". Somente uma instância de um nome de atributo é permitida dentro de uma tag inicial ou de uma tag vazia, ou seja, o exemplo a seguir seria inválido: <carta tipo="simples" tipo="comercial">.

A gramática formal da XML é apresentada na especificação XML do W3C pelo uso de notação na forma estendida de Backus-Naur, a EBNF (Extended Backus-Naur Form). Simplificadamente, a EBNF é um conjunto de regras formais que descrevem a sintaxe de uma determinada linguagem. Cada uma das regras é chamada de produção, e descreve um fragmento específico da linguagem. A validação se dá quando, através da aplicação das regras definidas para uma linguagem específica (XML, por exemplo), é possível reduzir um documento a uma única regra, sem que parte alguma do documento não tenha sido analisada.

Usando a EBNF, um programa que analise a estrutura da XML (comumente conhecido como *parser*) pode determinar que <remetente> José da Silva </remetente> está correto do ponto de vista da sintaxe da linguagem, enquanto (remente) José da Silva </remetente> não está. Existem programas que transformam automaticamente um conjunto de regras EBNF em um *parser*, tornando a EBNF uma opção bastante eficiente para representação de sintaxe de linguagens.

O exemplo 2, escrito em XML, significa a mesma coisa que o exemplo 1, escrito em HTML, visto que ainda podemos distinguir que se trata de uma carta. A diferença é que agora um programa de computador também seria capaz de distinguir, por exemplo, o que é cabeçalho, de quem e para quem é a mensagem, entre outras informações. Em outras palavras, a XML pode tanto adicionar informações sobre o conteúdo de um documento, como de sua estrutura.

O reconhecimento da estrutura do documento permite associar os dados que ele contém com um contexto específico, e utilizar as informações contidas na estrutura dentro deste contexto. Por este motivo, e também por ser possível definir os nomes das tags em XML, poderia se chegar à conclusão que a marcação define um significado para o elemento. Um elemento marcado como *<carta>*, seguindo este raciocínio, permitiria identificar o tipo de informação contida nele. Em outros termos, a marcação estaria definindo a semântica dos dados que ela representa.

No entanto, os *parsers* não fazem esta distinção. Eles cuidam apenas da verificação da sintaxe do documento, ou seja, da aderência do documento às regras de sintaxe da linguagem XML. O *parser* reconhece apenas que existe uma tag, cujo nome é carta. Se esse nome de tag trará algum significado para uma pessoa, depende inteiramente de um conhecimento prévio da pessoa sobre o significado daquele nome:

[...] O que transmite significado para um leitor humano não necessariamente faz a mesma coisa para uma máquina. Em XML, é o nome da tag que permite aos leitores humanos associarem um significado a um elemento, desde que o nome da tag combine com uma entrada no dicionário interno do cérebro do leitor. Se o nome da tag estiver fora do escopo de conhecimento do leitor, a tag não tem significado. Dois leitores com formações diferentes podem até mesmo interpretar a mesma tag como tendo dois significados diferentes (DAUM; MERTEN, 2000).

Assim, para definição do significado se faz necessária uma definição de regras semânticas formais: relações, restrições e definição de conceitos básicos sobre o significado

da noção que se pretende representar. Algumas aplicações de XML (linguagens desenvolvidas em XML para um contexto específico) se propõem a tratar disso, como, por exemplo, a Ontology Interchange Language (OIL). Esta aplicação da XML busca descrever de maneira formal, consensual e compartilhada um conjunto de conceitos que seja importante em um determinado domínio, tendo como objetivo tornar informações disponíveis de modo que seu significado – a semântica dessas informações – esteja numa representação acessível tanto para máquinas quanto para pessoas.

2.3 DEFINIÇÃO DA ESTRUTURA DE UM DOCUMENTO XML

A estrutura de um documento XML tem grande importância quando da escolha de qual forma de armazenamento será adotada para o documento. A seguir, serão mostradas as duas formas mais comuns para definição de estrutura e tipos de dados dos documentos XML, além de um breve comparativo entre essas formas.

2.3.1 *Document Type Definition (DTD)*

Uma Document Type Definition (Definição de Tipo do Documento, ou DTD) pode ser definida como um conjunto de regras que definem quais os tipos de dados e entidades que farão parte de um documento XML. Estas regras serão utilizadas para que o analisador sintático verifique se o documento está correto ou não. A DTD pode estar definida dentro do próprio arquivo XML ou em um arquivo à parte com extensão dtd, que deve ser incluído no código XML. A DTD pode ser usada para padronizar um documento XML e torná-lo bastante coerente com as necessidades do programador, porém sua criação não é obrigatória.

Abaixo está uma DTD para nosso exemplo de carta em XML:

```

<! DOCTYPE NOTA [
    <! ELEMENT carta (CABECALHO, PARAGRAFO*)
    <! ATTLIST carta tipo CDATA #REQUIRED>
    <! ELEMENT CABECALHO (DE, PARA)>
    <! ELEMENT DE (#PCDATA)>
    <! ELEMENT PARA (#PCDATA)>
    <! ELEMENT PARAGRAFO (#PCDATA)>
]>

```

Exemplo 3: DTD para o documento do exemplo 2.

2.3.2 XML Schemas Definition

A XML Schemas Definition (XSD) é uma recomendação do W3C criada no intuito de permitir que pessoas definam, através de regras, a estrutura, o conteúdo e a semântica de um documento XML. Segundo Obasanjo (2002, tradução nossa), “a recomendação de schema XML do W3C [...] fornece um meio sofisticado de descrever a estrutura e restrições no modelo de conteúdo de documentos XML”. A XSD tem a mesma função que DTDs, porém ela possui uma infinidade de novas características que a torna muito mais poderosa que DTDs.

Ela usa uma sintaxe baseada em XML para descrever as possíveis estruturas e elementos de um documento de XML. Permite também que sejam especificados tipos de dados, inclusive com verificação de seus valores limites. Por ser derivada da XML, é extensível e escalável. Por fim, suporta alguns conceitos da programação orientada a objetos, como herança e polimorfismo.

A seguir temos um exemplo de XSD para definir a estrutura do nosso exemplo de carta:

```

<schema xmlns=<http://www.w3c.org/2001/XMLSchema>>
<element name="carta">
  <complexType>
    <sequence>
      <element name="cabecalho" type="string">
        <complexType>
          <sequence>
            <element name="remetente" type="string" />
            <element name="destinatario" type="string" />
          </sequence>
        </complexType>
      </element>
      <element name="paragrafo" type="string" />
    </sequence>
    <attribute name="tipo" type="string" />
  </complexType>
</element>

```

xe
 mpl
 o 4:
 sch
 em
 a
 par
 a o
 doc
 um
 ent
 o

do exemplo 2.

O exemplo acima define um elemento complexo “carta” (que significa que o elemento carta pode ter elementos aninhados) que contém outro elemento complexo (“cabecalho”), mais um elemento simples “parágrafo” e um atributo “tipo” (o tipo da carta, social ou comercial). O elemento complexo “cabecalho” por sua vez, compreende dois elementos: “remetente” e “destinatario”.

DTDs possuem uma série de limitações que XSD não possui. Uma dessas limitações, que torna impossível o desenvolvimento de uma série de aplicações, é a falta quase completa de tipos de dados, especialmente para determinar o tipo de conteúdo de um elemento. Através de uma DTD não se pode dizer, por exemplo, que um elemento com o nome “preço” tem que conter um número, muito menos que ele deve ser inteiro, ou com dois dígitos decimais de precisão. Já com XSD pode-se definir estas características, além de ser possível a definição dos tipos para o conteúdo de seus elementos e atributos, como boolean, string, dentre outros.

O segundo problema é que DTDs têm uma sintaxe própria. Na verdade são necessários analisadores e Application Programming Interfaces (APIs, ou interfaces de

programação de aplicações) separados para controlar DTDs . Por exemplo, considerando a seguinte declaração de um elemento cujo nome seja TITULO: <!ELEMENT TITULO (#PCDATA)>. Pode-se notar que a sintaxe das DTDs não é a sintaxe XML. Já a linguagem XSD possui sua sintaxe baseada no XML, o que permite que se utilizem todos os recursos da linguagem XML também na linguagem XSD.

XSDs não substituem DTDs. Pode-se usar XSDs e DTDs em um mesmo documento. DTDs ainda fazem coisas que XSDs não podem fazer, como por exemplo declarar entidades. DTDs são amplamente utilizados, o que implica que analisadores e softwares continuarão aceitando DTDs enquanto eles aceitarem XML.

2.4 TECNOLOGIAS RELACIONADAS COM XML

A seguir, temos a definição de algumas tecnologias que são usadas em conjunto com a XML, para estender e ampliar suas possibilidades.

2.4.1 SAX

SAX (Simple API for XML, ou API simples para XML) não é um padrão do W3C, mas um desenvolvimento conjunto dos membros da lista de correspondência XML-DEV. Há diversos analisadores SAX para diferentes linguagens de programação, como Java, C++, Perl, e Delphi.

SAX é um analisador baseado em eventos – o analisador lê um fluxo de entrada XML e, no caso de eventos, o analisador chama métodos ou rotinas com nomes semelhantes no programa host, que poderão tomar a ação apropriada.

2.4.2 DOM

DOM (Document Object Model) é uma interface de programação de aplicação completa para documentos XML. Essa interface permite não apenas navegar dentro de documentos XML, mas também apanhar, acrescentar, modificar ou excluir elementos e conteúdo.

2.4.3 XPath

O XPath define como os nós dentro dos documentos XML podem ser acessados. Isso é feito com expressões de caminho que especificam um arquivo dentro de um diretório de arquivos. O XPath desempenha um papel fundamental em outros padrões, como XSL Transformations (XSLT) e XQuery, e também permite definir filtros dentro de uma expressão de caminho para restringir os caminhos de nós.

A localização de caminho é facilitada, pois isso será feito com expressões de caminho que especificam um arquivo dentro de um diretório de arquivos (DAUM; MERTEN, 2002). Por exemplo, para o documento XML do exemplo 2, a expressão que identifica o atributo “tipo” do elemento <carta> é dada por /carta/@tipo.

2.4.4 XSL

A XSL (Extensible Style Language) é uma linguagem para definir folhas de estilo para XML. Ela também aceita opções de impressão e transmissão de documentos XML entre várias plataformas. A especificação XSL contém três partes: a linguagem XSLT para transformação, XPath para acessar partes ou documentos XML inteiros e objetos de formatação XSL.

3. BANCOS DE DADOS E XML

Neste capítulo apresentamos os conceitos básicos de bancos de dados, além de abordarmos o modelo relacional e outras estruturas. Esses conceitos serão situados em um contexto histórico, traçando um perfil da evolução das tecnologias de bancos de dados até os bancos de dados nativos em XML.

3.1 DEFINIÇÃO DE BANCOS DE DADOS

Simplificadamente, um banco de dados é um conjunto de informações armazenados de tal forma que consigamos recuperá-los quando for preciso. Focando nos bancos de dados computacionais, uma definição mais precisa é dada por Date (2000): "sistema de manutenção de registros por computador (...) cujo objetivo global é manter as informações e torná-las disponíveis quando solicitadas".

Sistemas gerenciadores de bancos de dados, ou simplesmente sistemas de bancos de dados, são formados por um conjunto de dados associado a um conjunto de programas para acesso a esses dados, e têm por objetivo “garantir a segurança das informações armazenadas contra eventuais problemas com o sistema, além de impedir tentativas de acesso não autorizadas. Se os dados são compartilhados por diversos usuários, o sistema deve evitar a ocorrência de resultados anômalos” (SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S., 1999).

Os sistemas de bancos de dados mais robustos lidam com centenas de usuários concorrentes, oferecendo a eles transações com um tempo baixo de resposta e alta disponibilidade. O conceito de transação em banco de dados significa uma "unidade lógica de processamento de banco de dados que tem que ser completada em sua totalidade" (ELMASRI; NAVATHE, c2000) para garantir a correção desse processamento. Sistemas de

processamento de cartão de crédito e de informações bancárias são exemplos de aplicações que demandam por sistemas de bancos de dados robustos.

Para alcançar o grau desejado de desempenho, esses sistemas implementam algumas características que todas as transações terão que possuir. Tais características são comumente conhecidas como propriedades ACID. ACID é um acrônimo formado pelas iniciais de quatro características que os sistemas de bancos de dados precisam assegurar em suas transações: atomicidade, consistência, isolamento, e durabilidade.

- Atomicidade: ou uma transação é processada em sua totalidade ou então não é realizada.
- Consistência: se o banco de dados estava num estado consistente, ou seja, respeitando todas as restrições de seu esquema, ele deve permanecer consistente após a conclusão de uma transação.
- Isolamento: uma transação não pode interferir na execução de qualquer outra transação que esteja sendo executada concorrentemente.
- Durabilidade: as alterações feitas no banco de dados por uma transação devem ser duráveis, ou seja, não podem ser perdidas no caso de quaisquer falhas no sistema.

O grau de implementação das propriedades ACID para o controle de concorrência e recuperação de erros nas transações de um sistema de banco de dados serve como um índice de confiabilidade e segurança desse sistema.

Em poucas palavras, um sistema de banco de dados deve garantir segurança, integridade e eficiência, em um ambiente conveniente e eficiente para armazenamento e recuperação dos dados. Há algumas outras características importantes que são desejáveis em um sistema de banco de dados, como:

- Integração e compartilhamento dos dados;
- Suporte a múltiplas visões dos dados;

- Duplicação de dados controlada;
- Independência de dados;
- Representação mais próxima da percepção do usuário.

3.2 HISTÓRIA E EVOLUÇÃO DOS BANCOS DE DADOS

Podemos dividir os bancos de dados em 6 gerações distintas:

- 1ª Geração: caracterizada pelo processamento manual de dados, (papel e lápis).

A primeira geração tem início desde o início do surgimento da escrita até aproximadamente 1900. O conceito de bancos de dados não é novo: o surgimento da escrita, além de servir como forma de comunicação, veio de uma necessidade de armazenar dados para que pudessem ser lidos posteriormente, e mesmo antes esse tipo de armazenamento já era feito através de marcas (como por exemplo, para marcar um caminho de volta de uma caçada) ou mesmo através de desenhos.

- 2ª Geração: cartões perfurados e máquinas eletro-mecânicas para ordenar e tabular registros.

Por volta de 1800 Jacquard Lomm começou a produzir tecido a partir de padrões descritos em cartões perfurados. Em 1890, Hollerith usou a tecnologia dos cartões perfurados para fazer o censo demográfico dos EUA. Hollerith formou uma companhia para produzir máquinas que registravam dados em cartões, os ordenava e tabulava. Esta companhia se tornou a IBM. Até 1955, muitas companhias tinham andares inteiros para guardar cartões perfurados e processavam milhares de registros cada noite.

- 3ª Geração: uso de fitas magnéticas e de programas de computadores para executar

processamento batch de arquivos seqüenciais.

A terceira geração tem início nas décadas de 40/50, quando a UNIVAC desenvolveu uma fita magnética capaz de armazenar a informação de dez mil cartões. Com isso, surgiram os computadores que armazenavam programas. Assim surgiam as primeiras aplicações de negócios, como folhas de pagamento, porém o processamento era orientado a arquivo, sendo um grande arquivo lido seqüencialmente em batch, gravando-se os novos registros no arquivo principal. Essa geração perdura até meados da década de 70.

- 4ª Geração: uso de banco e dados e acesso online a dados.

Por volta de 1965, empresas precisavam de consultas a arquivos online, e não do processamento do dia anterior, como era o caso de sistemas de reservas de passagens, por exemplo. Assim surgem os primeiros bancos de dados de quarta geração, com suporte a transações online, que eram capazes de ler alguns registros, atualizá-los e retornar os novos valores ao usuário.

- 5ª Geração: acesso automatizado a banco de dados relacionais, processamento distribuído e cliente-servidor.

Em 1980 começa a ser aplicado o modelo relacional, com suas linguagens de definição e manipulação de dados, e sua álgebra capaz de efetuar operações com alguns registros, retornando um outro conjunto de registros. Surgem as aplicações Cliente/Servidor e também o processamento paralelo de bancos de dados, onde uma operação utiliza alguns dados de entradas, gera uma saída e essa saída é utilizada em outra operação.

- 6ª Geração: uso de novos tipos de dados, como documentos, imagens, som e vídeo.

A sexta geração tem início em 1995 e perdura até os dias de hoje. Com o sucesso da

orientação a objetos, começam a surgir os primeiros bancos de dados objeto-relacionais, capazes de armazenar objetos. Os bancos de dados precisam armazenar novos tipos de dados que vêm surgindo, como vídeo, som, imagem, documentos, páginas da Web e outros objetos complexos. Ao mesmo tempo, surge um novo formato para representação de dados: a XML.

Antes de detalharmos a relação entre a XML e os bancos de dados, cabe situarmos essa tecnologia na área de banco de dados dentre os modelos existentes.

3.3 TIPOS DE BANCOS DE DADOS

A seguir estão enumerados os principais e mais relevantes modelos para bancos de dados computacionais.

3.3.1 *Bancos de dados hierárquicos*

Um banco de dados hierárquico é composto por um conjunto de várias ocorrências de uma mesma estrutura, ou várias ocorrências de um único tipo de árvore. Assim, existe uma estrutura de registros compostos de diversos campos e estes são organizados em forma de árvore. Um registro pode ter um sub-registro e assim sucessivamente, de forma que estes são sempre unidos de tal sorte que um é o ascendente e o outro o descendente. Um ascendente pode ter vários descendentes, mas somente um ascendente. As relações entre registros são definidas por ponteiros.

3.3.2 *Bancos de dados de redes*

O modelo de banco de dados de redes é bastante parecido com o modelo hierárquico. Neste caso, os dados são representados por registros dispostos em forma de árvore, porém um

registro descendente pode ter qualquer número de ascendentes, inclusive zero. As relações entre os registros também são expressas por ligações, representadas por ponteiros.

3.3.3 Bancos de dados relacionais

Os bancos de dados relacionais consistem em um conjunto de tabelas, que contém linhas e colunas. Cada tabela possui várias colunas e cada uma das colunas tem um nome único. Essas tabelas podem possuir relacionamentos entre si. Toda consulta feita através de um operador resulta também em uma tabela.

A simplicidade do modelo, aliada a uma linguagem não-procedural poderosa, a Structured Query Language (SQL), “permitem ao desenvolvedor da aplicação especificar quais dados são necessários, e não como encontrar os dados. O sistema automaticamente encontra um caminho para as relações individuais, e decide como combiná-las com o objetivo de retornar o resultado desejado” (HAAS et al, 2002, tradução nossa).

Este é sem dúvida nenhuma o modelo lógico de dados mais utilizado na atualidade.

3.3.4 Bancos de dados orientados a objeto

Os bancos de dados orientados a objeto seguem um paradigma diferente dos outros tipos de bancos de dados. Estes não armazenam tabelas, nem árvores, mas sim estados de objetos. Utilizam linguagens orientadas a objetos e surgiu da necessidade da persistência de objetos, gerados pelos programas feitos utilizando linguagem orientada a objetos.

3.3.5 Bancos de dados objeto-relacionais

Os bancos de dados objeto-relacionais foram criados a partir da necessidade de se

estender o modelo relacional para lidar com aplicações novas, que fazem uso de dados não previstos inicialmente para serem representados por uma modelagem relacional. Como exemplos, podemos citar dados espaciais e geográficos em mapas, dados de poluição do ar ou água, imagens geradas por satélite para mapeamentos e previsão do tempo, dados complexos em projetos de engenharia ou nos estudos biológicos sobre genoma, dentre outros.

Para lidar com tais informações, os bancos de dados objeto-relacionais fazem uso de algumas das características das linguagens de programação e do modelo de dados orientados a objeto dentro do modelo relacional (ELMASRI; NAVATHE, c2000), como o suporte à herança de dados e de funções, suporte às rotinas definidas pelo usuário, representação de atributos multivalorados dentro de uma tabela e a presença de tipos abstratos de dados (ou ADTs, de Abstract Data Types).

Muitos dos novos tipos de dados podem ser modelados utilizando-se XML. Sistemas objeto-relacionais que incorporaram extensões para tornar possível o processamento e armazenamento de dados XML são conhecidos como “XML-aware” ou “XML-enabled”. Neste texto, será usada a expressão “sistemas de bancos de dados com suporte a XML”.

3.3.6 Bancos de dados nativos em XML

A XML dá ao mundo dos bancos de dados uma forma de resolver novos desafios, gerados principalmente pela grande expansão no uso da internet como meio para trocas comerciais e integração de processos entre organizações. Porém, em determinadas situações as soluções de bancos de dados até aqui citadas se mostram incompletas ou mesmo incompatíveis com o padrão XML.

Buscando conseguir armazenar dados XML sem perda de informação, de estrutura do documento, nem de desempenho no armazenamento e recuperação de informações, foi

desenvolvido um tipo específico de sistema de banco de dados: o sistema de bancos de dados nativo em XML.

Sistemas nativos em XML podem ser implementados sobre um modelo relacional, hierárquico, orientado a objetos, ou mesmo utilizar um formato proprietário de armazenamento. O que difere um banco de dados nativo em XML dos demais modelos até aqui apresentados é unidade de armazenamento: assim como em um banco de dados relacional uma linha na tabela é a menor unidade lógica de armazenamento, nos sistemas nativos em XML a menor unidade lógica é o próprio documento XML. Esse documento deve ser mantido intacto, e ao ser recuperado deve estar idêntico a quando foi armazenado.

Antes de uma análise mais detalhada sobre essa nova abordagem, é importante entender o que a XML traz de benefícios para o mundo dos bancos de dados, e qual lugar ela ocupa atualmente entre as tecnologias existentes.

3.4 XML, O NOVO PARADIGMA

A ampla disseminação do uso da internet aconteceu em cima do uso de padrões abertos, ou seja, não-proprietários: código-fonte, interfaces, sistemas operacionais etc., independentes de um fabricante específico e abertos a alterações, adequações e extensões que se fizessem necessárias, sempre mantendo o alinhamento com as regras definidas para estes padrões.

Essa disseminação trouxe novas maneiras para o aproveitamento comercial da internet, destacando-se dentre elas o comércio eletrônico entre empresas (business-to-business, ou B2B), entre empresas e órgãos governamentais (business-to-government, ou B2G), e entre empresas e clientes finais (business-to-consumer, ou B2C). A importância do comércio eletrônico pode ser expresso por números: de acordo com uma pesquisa recente, 125 bilhões e 800 milhões de reais foram movimentados em operações de B2B, B2C e em investimentos

nos bancos e corretoras online no período de um ano, apenas no Brasil (Os Reis do E-commerce. Info Exame, abr. 2003).

Para a viabilidade técnica do comércio eletrônico, deve haver uma forma de integração eletrônica de dados entre os diferentes envolvidos no processo comercial. Uma forma de resolver este tipo de situação seria a transmissão de dados em um formato conhecido pelos parceiros comerciais, de forma que um entenda o significado dos dados enviados pelo outro. Até pouco tempo atrás, o que estava disponível nesse sentido era a tecnologia de troca eletrônica de dados, ou Electronic Data Interchange (EDI). Resumidamente, a EDI “define uma plataforma técnica, baseada num conjunto de padrões estruturados para troca de informações entre participantes num mercado” (ALBERTIN, 2001). Ainda que tenha viabilizado as primeiras iniciativas em comércio eletrônico, as implementações de EDI têm algumas limitações que impediram sua adoção de forma mais generalizada:

- Geralmente requer uma rede de conexão, privada e cara, entre os parceiros de negócios;
- Os parceiros precisam ter um relacionamento preexistente, para dar a eles tempo para estabelecer uma rede;
- A EDI não é interativa, inibindo a possibilidade de negociação e discussão dinâmica.

Com formas de acesso baratas, preços não dependentes da quantidade de informação utilizada, padrões abertos e amplamente adotados, e o desenvolvimento de técnicas de segurança, a internet se firmou como opção a essa situação de altos custos, requerimentos rígidos e soluções fechadas que inibia a entrada de mais participantes no mundo do comércio eletrônico. Além disso, o crescimento vertiginoso da adoção da internet, pelos motivos já citados, propiciou condições para a entrada de novos parceiros de negócio no mundo do comércio eletrônico. Tornou-se possível o relacionamento comercial "um-para-muitos" em substituição ao "um-para-um" de até então. Soluções fechadas como a EDI diminuía o

desenvolvimento e adoção do comércio eletrônico, uma vez que estabeleciam uma situação desfavorável às trocas comerciais.

A XML se enquadra nesse cenário de busca por padrões abertos que tiram o máximo de vantagem da arquitetura da internet. Redes de computadores tratam basicamente da troca de informações, e a internet nada mais é que uma rede de computadores. Com a XML, essas trocas de informações passaram a ter um padrão "aberto" para o intercâmbio de dados.

Outra característica importante da XML é a sua portabilidade. Dizer que ela é portátil equivale a dizer que pode ser usada sem adaptações entre plataformas diferentes de hardware e software. A razão para tanto é que qualquer documento escrito usando o padrão XML será sempre um arquivo de texto simples, que usa as regras de marcação da XML. Assim, este arquivo texto pode ser trocado entre diferentes arquiteturas sem ser necessário sofrer alterações em seu formato: arquivos de texto podem ser reconhecidos e processados por virtualmente qualquer plataforma.

Ao considerarmos o uso da XML como forma para o armazenamento em bancos de dados, devemos levar em conta o uso não somente do padrão XML em si, mas também de todas as tecnologias que o envolvem como XSL, XSD, linguagens para consulta como XPath, Xquery e XQL, e analisadores DOM e SAX, dentre outras. O uso combinado destas tecnologias, muitas das quais padrões publicados ou em fase de aceitação pelo W3C, é que torna viável o uso da XML como padrão robusto de intercâmbio e armazenamento de informações. Sem as tecnologias que o englobam, a XML é apenas mais uma linguagem de marcação, sem o poder e eficiência que apresenta quando usada em conjunto com elas.

A adoção da XML vem num ritmo crescente por parte daqueles que usam a internet como meio para a troca de informações, mas é interessante observar que sua importância não se limita ao intercâmbio de dados. Ao contrário: ela invade a área das aplicações de bancos de dados, e com grande destaque. Kroenke (2000, tradução nossa) chega a afirmar que “a XML

pode ser o desenvolvimento mais importante para aplicações de bancos de dados desde o modelo relacional”. Alguns motivos vêm impulsionando a utilização desta nova tecnologia na área de bancos de dados.

3.5 CARACTERÍSTICAS DA XML PARA BANCOS DE DADOS

A XML tem características que a aproxima do mundo dos bancos de dados, e que têm impulsionado sua adoção nessa área específica. Vamos analisar as principais delas a seguir.

3.5.1 *Desenvolvimento de padrões de uso específico*

Uma característica importante da XML é a possibilidade de que grupos da indústria criem padrões com ampla aceitação para diferentes nichos como, por exemplo, a indústria de publicação de conteúdo, financeira, de intercâmbio de dados, dentre muitas outras. Todos estes padrões são desenvolvidos em cima da linguagem XML, mantendo as características originais da linguagem e adaptando-a para um uso mais específico.

A XML é um padrão proposto pelo W3C, e que tem atualmente ampla aceitação de uso. Grandes empresas como Microsoft, IBM, BEA Systems, e Oracle apóiam esse padrão e têm aplicações comerciais que fazem uso de XML. A partir da adoção mais ampla da XML, foram surgindo novas necessidades que precisavam ser atendidas. Assim, novos padrões baseados em XML foram criados: XPath, SAX, (Simple API for XML), DOM (Document Object Model), XQuery, dentre outros. Cada um deles ocupa-se de uma necessidade específica que não foi atendida diretamente pela XML. A vantagem dessa abordagem é que para cada necessidade que surja na utilização da XML pode ser desenvolvida uma solução mais específica, sem fazer com que o padrão em si torne-se muito complexo, contrário, portanto, ao conceito primordial da XML.

3.5.2 Padrão para visões de bancos de dados

Uma visão é "qualquer relação que não faz parte do modelo lógico, mas é visível para o usuário como uma relação virtual" (SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S., 1999). Visões são implementadas para resolver questões de segurança e desempenho. Elas simplificam consultas ao banco de dados, ao restringir os resultados das consultas ao conjunto de atributos procurados, e permitem a criação de um conjunto de relações personalizadas com base em questões de segurança, fazendo com que alguns dados não estejam disponíveis para certos usuários.

Vamos analisar um exemplo prático de uso de visões de dados. Suponhamos que uma tabela com informações sobre funcionários tenha os seguintes atributos: nome completo, cargo, data de admissão, salário, números de documentos pessoais, número de identificação dentro da empresa, ramal e filial na qual trabalha. Para disponibilizar as informações de contato deste funcionário – nome, ramal e filial, por exemplo – para consulta de todos na empresa, cria-se uma visão desta tabela com os atributos desejados. Assim, evita-se a exposição de informações sensíveis como números de documentos pessoais ou o valor do salário. Também diminuirá bastante a massa de dados com a qual as consultas terão que lidar, o que traz benefícios em termos de desempenho.

A XML fornece um meio padronizado para expressar a estrutura de visões de dados. Essa característica da XML é importante porque fornece a aplicações de bancos de dados que sejam capazes de interpretar uma DTD (ou outra forma de definição da estrutura de um documento XML) a possibilidade de transmitir e processar visões arbitrárias do banco de dados. O que antes demandava o desenvolvimento de um protocolo proprietário que especificasse a estrutura da visão, agora pode ser alcançado pela interpretação de uma DTD.

3.5.3 Separação entre estrutura, conteúdo e materialização dos dados

A separação clara entre estrutura e conteúdo é um dos pontos principais da XML, e também se destaca quando se pensa em XML como formato de armazenamento de dados. Vamos voltar ao exemplo da carta. Esse exemplo simplesmente populou uma estrutura em XML de um documento com os dados reais de uma carta. Suponhamos que esse conteúdo fosse publicado em XML com informações vindas de um banco de dados. Poderíamos ter diversas cartas, de diferentes bancos de dados, usando a mesma estrutura. Agora imagine que possamos, já que sabemos quais as tags que delimitam a estrutura do documento, definir um outro arquivo que diz que tudo que estiver entre as tags <cabeçalho> utilize fonte “Verdana” de tamanho 2, e o que vem entre as tags <paragrafo> utilize tamanho 1. Poderíamos ainda dizer que o conteúdo das tags <de> deveria ser mostrado em vermelho e o conteúdo das tags <para> deveria ser mostrado em verde. Seria mais interessante ainda podermos ter mais de um arquivo desses, que utilizassem a mesma origem de dados, mas que exibissem o resultado de forma diferente. Isso tudo é possível utilizando-se folhas de estilo CSS ou XSL junto com a XML.

Além disso, múltiplos documentos XML podem fazer uso da mesma DTD, que ficaria armazenada em um arquivo separado dos demais documentos XML. Damos como exemplo da vantagem que isso traz, o caso de uma empresa cujo banco de dados esteja disponível através da internet. Uma DTD para seus documentos XML poderia ser publicada num site na Web, e outras empresas em qualquer lugar do mundo podem desenvolver aplicações de bancos de dados que criem documentos XML em conformidade com esta DTD. Torna-se irrelevante a forma como os dados foram criados. Desde que estejam de acordo com a DTD publicada, eles são válidos para aquele banco de dados.

4. ARMAZENAMENTO DE DADOS XML

Existem diversas alternativas para armazenamento de dados XML. É importante compreender a quais problemas cada solução se encaixa melhor, pois isso torna possível escolher a alternativa mais apropriada para cada situação. Este capítulo aborda um estudo da estruturação das informações em documentos XML, para em seguida expor as soluções existentes para persistência de XML.

4.1 A XML NO CONTEXTO ATUAL DOS BANCOS DE DADOS

Já existe no mercado um sem-número de sistemas de bancos de dados que lidam de forma plenamente satisfatória com dados relacionais (ou seja, dados que estejam no modelo lógico relacional, com tabelas, atributos e registros), predominantes no contexto empresarial. Porém, após o surgimento e fortalecimento do padrão XML para intercâmbio e armazenamento de dados, surgiram novas necessidades com as quais os desenvolvedores de sistemas de bancos de dados relacionais tiveram que lidar.

De modo geral, ao perceberem a importância que o padrão XML assumiu, os grandes fornecedores de sistemas de bancos de dados adicionaram extensões para XML aplicadas a seus sistemas gerenciadores de bancos de dados relacionais, com o intuito de adaptar estes sistemas para que pudessem de alguma forma armazenar dados XML em suas tabelas relacionais. Podemos relacionar como principais fornecedores de soluções para bancos de dados relacionais a Microsoft, a Oracle e a IBM.

Outra abordagem foi o desenvolvimento de um novo tipo de sistema de bancos de dados, voltados especificamente para lidar com o formato XML, que vieram a serem conhecidos como sistemas de bancos de dados nativos em XML.

Antes da decisão sobre qual solução adotar, ou mesmo sobre a necessidade de se

armazenar os dados que são produzidos e/ou transportados no formato XML, deve-se analisar cuidadosamente outro fator fundamental, sobre o qual as soluções disponíveis estão baseadas. Este fator é o grau de estruturação das informações em questão.

4.2 GRAU DE ESTRUTURAÇÃO

A forma como as informações estão estruturadas em um arquivo XML permite classificá-las em duas categorias: informações voltadas a dados ou informações voltadas para documentos. Na literatura sobre o tema são encontrados para essas categorias, respectivamente, os termos "data-centric documents" e "document-centric documents" (Neste trabalho optou-se por traduções mais livres, visando atingir maior clareza).

4.2.1 *Informações voltadas para dados*

Informações voltadas para dados se enquadram em uma estrutura bastante regular, que oferece uma granularidade mais fina para os dados. Por granularidade, entende-se que representa a menor unidade independente de dados que a estrutura permite representar. Desse modo, informações voltadas a dados permitem que se lide com os dados em XML no nível dos seus elementos. A ordem na qual esses elementos serão armazenados não é importante, exceto quando algum *parser* estiver validando a estrutura do documento XML contra uma DTD ou XSD qualquer. Informações oriundas de tabelas ou cujo destino seja uma tabela de banco de dados relacional podem quase que certamente ser classificadas como voltadas para dados. Informações que se encaixam nesta categoria simplesmente fazem uso das características de portabilidade da XML para o transporte de dados entre aplicações de bancos de dados. Elas não serão analisadas ou mesmo lidas diretamente por pessoas: mais comumente, serão transmitidas de um banco de dados para outro, através de aplicações

usando muito provavelmente a internet como meio de troca dos dados.

Segue um exemplo simples de um documento XML, com informações voltadas para dados:

```
<lembrete>
  <horário>14h30</horário>
  <local>bar do Nirtão</local>
  <assunto>
    Discutir a conjuntura política do mundo globalizado.
  </assunto>
</lembrete>
```

Exemplo 1: XML voltado para dados.

4.2.2 Informações voltadas para documentos

As informações voltadas para documentos são ditas como sendo semi-estruturadas, ou seja, elas têm uma estrutura menos rígida. Em outros termos, como explicado por Vianu (2001, tradução nossa), esses dados “não mais se adequam bem em tabelas. Ao contrário, eles são autodescritíveis e irregulares, com pouca distinção entre o esquema e os dados”. A granularidade obtida é menor que a das informações voltadas para dados: a menor unidade independente de dados é geralmente o próprio documento XML como um todo. Conforme Nambiar et al (2002, tradução nossa), “o uso voltado para documentos da XML depende não apenas da representação dos dados expressa através de marcações, mas também da ordenação dos componentes dos dados”. Essa ordenação fatalmente seria perdida se os dados fossem modelados para uma tabela relacional.

Informações voltadas para documentos são produzidas diretamente por pessoas (em oposição a terem sido geradas por algum processamento computacional prévio) e serão consumidas também por pessoas (ao invés de apenas por outras aplicações).

Esses documentos aproveitam outras possibilidades oferecidas pela XML, além da portabilidade, como a representação de conteúdo que não obedece a estruturas rígidas. Fica

difícil imaginar um documento XML contendo uma mensagem de correio eletrônico, ou um manual técnico, ou mesmo o texto de um anúncio, e que fosse modelado satisfatoriamente em tabelas de atributos atômicos de um sistema de bancos de dados relacional.

Um exemplo de documento XML com informações voltadas para documento pode ser dado por:

```
<lembrete> Não se esqueça da nossa reunião lá no <local>bar do  
Nirtão</local>, às <horário>14h30</horário> mais ou menos,  
para podermos <assunto> discutir a conjuntura política do  
mundo globalizado </assunto>. Vai ser demais!  
</lembrete>
```

Exemplo 2: XML voltado para documento.

4.2.3 Mas porque usar dados semi-estruturados?

Uma vez que a imensa maioria dos sistemas de bancos de dados trabalha com o modelo relacional, e realiza as tarefas que deles se espera com um grau satisfatório de segurança e confiabilidade, pode surgir a pergunta: qual a importância dos dados não-relacionais?

Uma resposta bastante evidente a esta pergunta é a World Wide Web. Ela é uma coleção de conteúdo de diversas fontes, numa evolução livre de regras e sempre em mutação, ao contrário dos sistemas de bancos de dados, que de acordo com Vianu (2001, tradução nossa), “impõem um estrutura rígida, e disponibilizam consultas, atualizações, bem como transações, concorrência, integridade, e recuperação em caso de falhas, num ambiente controlado”. Ainda de acordo com este autor, a Web tem papel destacado no campo dos bancos de dados:

[...] há uma necessidade tremenda de funcionalidades de bancos de dados para fornecer e acessar dados eficientemente na Web e para uma ampla gama de aplicações. E, a despeito das diferenças, o que se tira é que o conhecimento da área de bancos de dados permanece sendo

extremamente valorizado e efetivo. [...] O uso do paradigma de bancos de dados na Web é uma história de sucesso, um testemunho da robustez dos bancos de dados como um campo de aplicação (VIANU, 2001, tradução nossa).

A XML, como alternativa portátil e expansível de representação, armazenamento e manipulação de conteúdo, desponta para uso em conjunto com a Web, tornando viáveis sistemas para publicação, navegação e interpretação das informações presentes nela. Sistemas específicos para armazenamento e gerenciamento de conteúdo, construídos sobre as tecnologias de bancos de dados e XML, são exemplos de aplicação deste conceito na prática.

4.2.4 Dados versus documentos em bancos de dados

Informações em XML voltadas para dados são na grande maioria das vezes armazenadas em bancos de dados que seguem os modelos mais tradicionais, como o hierárquico, orientado a objetos ou (o caso mais comum) em sistemas de bancos de dados relacionais. Esse tipo de solução pode ser atendido de duas formas. A primeira, por produtos de middleware entre os bancos de dados e as aplicações que os utilizam. A segunda forma de abordagem para esta situação seria o uso das extensões oferecidas pelos sistemas bancos de dados relacionais, que permitem que dados XML sejam armazenados dentro das tabelas relacionais.

Ferramentas de bancos de dados que lidam com informações voltadas a dados podem representar eficientemente apenas os dados estruturados e relacionais. Tipos de dados complexos, semi-estruturados ou mesmo sem estrutura definida não têm uma representação adequada no modelo relacional. Para representar qualquer tipo de conteúdo em XML, desde dados relacionais até dados multimídia e tipos de dados complexos, passando por dados semi-estruturados, a opção que atinge os objetivos propostos seria o armazenamento de informações voltadas para documentos.

Fazendo uso da XML, estruturas de dados não-relacionais passam a contar com uma forma organizada de armazenamento, que permite a representação de estruturas complexas como árvores ou grafos. Alguns exemplos de soluções que utilizam esta abordagem voltada a documento são sistemas de gerenciamento de conteúdo e servidores de dados XML. Estas soluções têm algo em comum: são construídas sobre bancos de dados nativos em XML.

Característica	XML para dados	XML para documentos
Estruturação	Regular	Irregular / inexistente
Estruturas representadas	Relacionais	Complexas (árvores, grafos, etc)
Granularidade	Linhas da tabela	O próprio documento XML
Ordem dos elementos	Não é mantida	Tem que ser mantida
Uso das informações	Aplicativos	Aplicativos / pessoas

Tabela 1: comparativo entre XML para dados e XML para documentos

Essa categorização de informações voltadas a dados ou a documentos não fica sempre clara para todos os documentos XML. Um documento XML altamente estruturado, como uma fatura ou nota fiscal, pode conter conteúdo semi-estruturado (uma descrição mais detalhada de um item qualquer desta nota, por exemplo). Por outro lado, mesmo documentos semi-estruturados, mais voltados para consumo humano, como um livro ou manual técnico, podem conter informações com maior grau de estruturação (nome do autor, data da última revisão, ou outros metadados – dados a respeito dos dados). Ainda assim, a partir de uma análise do grau de estruturação das informações em XML, é possível apontar a solução mais indicada para o armazenamento e recuperação dessas informações.

4.3 SOLUÇÕES PARA INFORMAÇÕES VOLTADAS A DADOS

Como dissemos, informações em XML voltadas para dados são na grande maioria das

vezes armazenadas em sistemas de bancos de dados relacionais, e isto pode ser feito através de aplicações de middleware ou por extensões de suporte para XML adicionadas aos sistemas de bancos de dados relacionais.

Como regra geral, para este tipo de estruturação dos dados serão usados bancos de dados relacionais. Tecnologias de suporte à XML nativa também poderão ser usadas, porém estarão limitadas a servir como gerenciador de entrada e saída de dados XML, sem fazer uso de suas características voltadas especificamente para a manipulação de XML conforme será abordado mais adiante.

4.3.1 Middleware de conversão

Produtos de middleware para bancos de dados são aplicações que servem como camada intermediária entre os sistemas de bancos de dados relacionais e aplicações que necessitem utilizar dados XML. Eles trabalham como conversores, extraíndo dados (na maior parte das vezes no modelo relacional de banco de dados), transformando-os para o formato XML, e passando então os dados convertidos – provavelmente através da internet – para a aplicação XML que manipulará esses dados. No sentido inverso, ao receber de uma aplicação os dados XML a serem gravados no banco de dados, essa aplicação de middleware converte-os para o formato de tabelas do banco de dados que pode então receber e gravar estes dados. Nesse caso, a XML está servindo apenas como meio de transporte entre sistemas relacionais ou entre um repositório XML e um banco de dados relacional.

4.3.2 Extensões em bancos de dados objeto-relacionais.

Fazendo uso das extensões para XML presentes atualmente em sistemas gerenciadores de bancos de dados objeto-relacionais, é possível atingir algumas das vantagens oferecidas

pelo uso da XML em bancos de dados, sem abrir mão do nível de confiabilidade e segurança que os sistemas relacionais apresentam atualmente para gerenciamento de dados relacionais. É possível criar visões em XML a partir de dados relacionais, executar consultas SQL geradas a partir da tradução de pesquisas em XML, além de mapear os dados de XML para relacionais e poder publicá-los como XML a partir de uma conversão posterior.

Um exemplo de mapeamento de XML para uma tabela relacional está representado a seguir.

```
<Funcionario>
  <Func>
    <Nome> Márcia Silva </Nome>
    <Cargo> Psicóloga </Cargo>
    <Salario> 1350,00 </Salario>
  </Func>
  <Func>
    <Nome> André Nunes </Nome>
    <Cargo> Geólogo </Cargo>
    <Salario> 1150,00 </Salario>
  </Func>
</Funcionario>
```

Funcionario

Nome	Cargo	Salario
Márcia Silva	Psicóloga	1350,00
André Nunes	Geólogo	1150,00

Exemplo 3: trecho de documento XML e o resultado de seu mapeamento para tabela.

Primeiro, temos o documento XML a ser mapeado. Observe que é um documento com uma estrutura altamente regular. Para cada elemento mais interno será criado um atributo na tabela, cujo valor passará a ser o conteúdo deste elemento. O elemento raiz, <Funcionario>, serve como limite do que deve ser mapeado e seu nome será representará a nova tabela gerada.

4.4 SOLUÇÕES PARA INFORMAÇÕES VOLTADAS A DOCUMENTOS

O uso de bancos de dados com suporte a XML consegue suprir muitas necessidades, mas tem também limitações. Apesar de colocar dados XML ao alcance dos sistemas de bancos de dados relacionais, nem todas as tecnologias que giram ao redor da XML são

suportadas pelas extensões de XML desses sistemas.

Para a manipulação de estruturas voltadas para dados, ainda que isso seja uma limitação, pode não representar um impedimento. Já para as estruturas de dados XML voltadas para documentos, pode não ser a saída ideal (e geralmente não é) abrir mão de certas características presentes nos documentos. Por exemplo: após passar por um mapeamento simples de XML para tabelas relacionais, não é mais possível recuperar a ordem na qual os elementos aparecem no documento XML original – vale lembrar que numa estrutura relacional, informações sobre a ordem dos atributos não são levadas em consideração. Sem essa ordenação, perde-se a estrutura implícita dos dados dentro da estrutura geral do documento.

Assim, para informações em XML que sejam voltadas a documentos, temos que buscar outras abordagens para seu armazenamento e manipulação sem perda de funcionalidade nem de informações estruturais.

4.4.1 Arquivo XML como repositório de dados

Relacionais ou não, dados podem ser representados de forma estruturada usando a XML. Uma vez que há um conjunto de dados dentro de cada documento, os dados estão organizados dentro de uma estrutura e podem ser recuperados a partir do documento, poderia se chegar à conclusão que um documento XML em si possa ser um banco de dados. Uma análise mais cuidadosa, no entanto, permite antever possíveis resultados: arquivos imensos com centenas, talvez milhares de documentos dentro dele, todos devidamente estruturados em suas tags.

Apesar de estar de acordo com a definição de bancos de dados (no sentido mais estrito do termo), claramente essa não é uma opção para armazenamento e recuperação dos dados, uma vez que na prática torna a manipulação dos dados inviável. Basta imaginar as

dificuldades na gravação, consulta e recuperação de um arquivo único com um tamanho muito grande, o que não é raro em sistemas de bancos de dados corporativos. Cada operação no conjunto de dados exigiria uma quantidade muito grande de operações de entrada/saída, e o uso de índices para este arquivo poderia na melhor das hipóteses atenuar o problema, mas não solucioná-lo. Outra questão seria a dificuldade, ou mesmo impossibilidade em alguns casos, de serem implementadas características importantes para um sistema de banco de dados, como o gerenciamento de múltiplos acessos concorrentes aos dados, a manutenção da integridade física e lógica da base de dados, dentre outros.

4.4.2 Uso do sistema de arquivos

A forma adotada para armazenamento e recuperação de documentos XML de modo mais simples possível é a seguinte: eles são gravados no próprio sistema de arquivos do computador. Essa forma de armazenamento tem uma “granularidade grossa”, onde a menor unidade de dados recuperável é o próprio documento. Um exemplo de granularidade mais “fina” seria uma tabela de preços em um banco de dados relacional, onde a menor unidade de dados recuperável seria uma linha (ou tupla, ou registro) ao invés da tabela toda. Após os documentos serem gravados como arquivos no sistema de arquivos do sistema operacional, ferramentas de busca de texto do sistema operacional podem ser então utilizadas para pesquisas dentro do documento. O próprio sistema de arquivos se encarrega de garantir a integridade dos dados armazenados.

Essa abordagem funciona com eficiência em situações nas quais o volume de documentos é pequeno e o número de usuários desses documentos é reduzido, quando as exigências de integridade dos dados e segurança não são essenciais, e se acessos de vários usuários simultaneamente aos dados não forem uma realidade. Sendo assim, para cenários

corporativos, com necessidades mais robustas, devem entrar em cena soluções mais eficientes: os sistemas de bancos de dados.

4.4.3 Sistemas relacionais para armazenamento de documentos XML

Conforme discutido anteriormente, os sistemas de bancos de dados relacionais possuem algumas características que os tornam a opção preferida para aplicações corporativas: processamento eficiente de grandes volumes de transações, suporte para múltiplos acessos concorrentes aos dados e segurança implementada com o uso de visões de dados.

Para armazenar dados XML em sistemas de bancos de dados relacionais, pode-se fazer uso de atributos do tipo texto. Neste caso, um documento XML seria armazenado por inteiro dentro de um atributo de tipo texto. Porém, ainda que conservando a ordenação implícita que as informações XML voltadas a documento precisam, buscas nesses dados se tornariam inviáveis: não há ferramentas adequadas em sistemas relacionais para diferenciar os dados da marcação, por exemplo.

Em sistemas objeto-relacionais, existem tipos de atributos para armazenar informações sem considerar a estrutura, como BLOB (Binary Large Object) e CLOB (Character Large Object). Sendo armazenado por inteiro, um documento XML não sofre a perda de nenhuma informação implícita que o mapeamento para estruturas relacionais acaba acarretando. É possível até mesmo implementar uma indexação específica para estes documentos, conforme descrito por Bourret (2003): numa tabela (a tabela dos documentos) ficam armazenadas duas informações: o documento XML (dentro de um atributo do tipo CLOB, por exemplo) e um campo de identificação, possivelmente numérico, para aquele documento em específico dentro da tabela, ou seja, um campo chave. Numa outra tabela, a de indexação, ficam

guardadas duas informações: o campo a ser indexado (qualquer campo que possa ser palavra-chave em uma busca, como placa ou número do chassi no caso de dados sobre veículos, por exemplo) e o campo de identificação usado aqui como chave estrangeira, servindo para indicar a entrada do documento procurado na tabela de documentos.

A desvantagem no uso de CLOBs, BLOBs ou mesmo de atributos do tipo texto para armazenamento de documentos XML é que uma vez que o documento seja armazenado, a recuperação do documento (ou de trechos dele) torna-se muito mais difícil. Para alterar a informação de dado elemento de um documento XML armazenado em um CLOB, seria necessário recuperar o documento todo, ler e alterar a informação desejada e depois gravar o documento todo de volta no campo CLOB, valendo o mesmo para consultas nestes dados.

4.4.4 Sistemas de bancos de dados nativos em XML

Uma definição abrangente para os sistemas de bancos de dados nativos em XML foi elaborada pelo XML:DB Initiative, uma iniciativa da indústria que reúne um grupo de empresas com interesses em XML e bancos de dados XML. Segundo esse grupo, um sistema nativo em XML é um sistema de banco de dados que “[...] define um modelo (lógico) para um documento XML - oposto ao dado naquele documento - e armazena e recupera documentos de acordo com este modelo. No mínimo, o modelo tem que incluir elementos, atributos, PCDATA, e a ordem do documento” (XML DB INITIATIVE, c2000-2003, tradução nossa).

A definição deste grupo da indústria também especifica outras características presentes em um banco de dados nativo em XML. Uma delas é sobre a granularidade do banco de dados, que deve ter como unidade fundamental de armazenamento lógico um documento XML, em comparação com um registro como unidade fundamental de armazenamento lógico num banco de dados relacional. Diz também que a implementação deste modelo de armazenamento lógico é independente do modelo físico de armazenamento que venha a ser

adotado (os dados podem ser armazenados compactados, indexados, ou de outra forma que venha a ser desenvolvida). Todas as informações de marcação e sobre a ordem implícita dos elementos dos documentos XML, de acordo com esta definição, devem estar armazenadas no banco de dados.

Um banco de dados nativo em XML preserva a estrutura física do documento: comentários, instruções de programação, DTDs, e assim por diante. Para serem considerados nativos em XML, os bancos de dados precisam fazer valer o conceito de “round-tripping”, que significa que “armazenar um documento XML não irá corrompê-lo. Isto é, se você armazenar um documento XML no banco de dados e então mais tarde recuperar aquele documento, você terá de volta exatamente o que armazenou” (MCGOVERAN, 2001, tradução nossa).

Também é interessante observar que os sistemas nativos em XML devem ser capazes de fazer o armazenamento, manipulação e recuperação de documentos XML sem definições explícitas de sua estrutura (sem fazer uso de uma DTD ou XSD, por exemplo). Já em sistemas de bancos de dados com suporte para XML, as técnicas utilizadas para tanto são particularmente limitadas.

Estes sistemas nativos em XML necessitam fazer uso de XPath, DOM, ou APIs similares relacionadas à XML para o acesso a dados. Usando estes métodos, é possível buscar e recuperar uma informação específica dentro de um documento ou mesmo uma coleção de documentos relacionados. Já os sistemas relacionais com suporte para XML oferecem acesso aos dados através de APIs como a Open DataBase Connectivity (ODBC).

O conceito de coleção de documentos XML fornece um equivalente às tabelas do modelo relacional: coleções são conjuntos de documentos relacionados entre si. Por exemplo, se notas fiscais fossem armazenadas em um banco de dados nativo em XML, poderiam ser agrupadas por critérios específicos como coleções por produto, ou por vendedor.

No que diz respeito às características negativas, os bancos de dados nativos em XML não são adequados às tarefas de agregação. Funções de agregação, de acordo com Silberschatz, Korth e Sudarshan (2001), “[...] são aquelas que, quando aplicadas, tomam uma coleção de valores e retornam um valor simples como resultado”. Exemplos de funções de agregação são as funções de soma ou de média, que tomam um conjunto de valores e devolvem um único valor como saída de sua operação. Quanto a essa característica, a crítica feita por Willians (2001, tradução nossa) ao armazenamento de dados como XML deve ser considerada:

Uma vez que a informação é persistida no nível do documento ou então ‘quebrada’ genericamente em nós, agrupar informação e ‘dobrá-la’ (em somas, médias, e assim por diante) é bastante difícil, necessitando codificação adicional na camada intermediária.

Além deste fato, deve-se levar em consideração também que os dados em XML têm uma estrutura mais complexa que os relacionais. Se não for realmente necessário o armazenamento de dados na forma de documentos completos (ao invés de elementos XML sendo decompostos como atributos de tabelas relacionais), um sistema nativo em XML pode não ser – via de regra, não é – a saída mais adequada.

Por fim, a maioria dos sistemas gerenciadores de bancos de dados nativos em XML não está ainda num estágio de amadurecimento comparável com os grandes gerenciadores relacionais do mercado. Nem todos eles oferecem suporte para garantia das propriedades ACID das transações e, daqueles que oferecem esse suporte, nem todos o fazem de forma completa. A tendência, entretanto, parece ser a da evolução, tanto do nível de suporte para XML embutido nos sistemas relacionais quanto no que diz respeito às características de robustez e segurança dos sistemas nativos em XML.

5. XML EM SISTEMAS DE BANCOS DE DADOS COMERCIAIS

Este capítulo apresenta uma visão geral de como os principais bancos de dados relacionais, aplicativos de middleware e sistemas de bancos de dados nativos em XML do mercado implementam suporte para dados XML.

5.1 MIDDLEWARE

5.1.1 *DB2XML*

DB2XML desenvolvido por Volker Turau, de acordo com o site oficial [<http://www.informatik.fh-wiesbaden.de/~turau/DB2XML/>] é um tipo Classes Java para transferência de dados de um banco de dados relacionais para um documento XML. Essas Classes podem ser usadas em um aplicativo comum ou em um "servlet". O modelo do documento XML do produto é um conjunto de tabelas, para que o usuário especifique 1 ou mais SELECTs.

5.1.2 *BeanStalk*

Beanstalk da Empresa Beanstalk Corp é um mecanismo de banco de dados relacionais que faz a ligação entre o aplicativo e o banco de dados, utiliza "object sets messaging model" para aumentar a performance das JOINS. Em uma implementação, Adicionando múltiplas tabelas como se fossem 2 tabelas, criando tabelas temporárias de um resultado, adicionando esta com a próxima tabela.

Beanstalk utiliza ODBC para se comunicar com o banco de dados. Clientes podem utilizar JDBC para se comunicar com o Beanstalk.. Mais informação podem ser obtidas no site oficial do Beanstalk [<http://www.beanstalk.bz/Products/products.html>].

5.1.3 Allora

Allora desenvolvido pela HiT Software [www.hitsw.com] é um "middleware" que o aplicativo chama para transferir dados entre um documento XML e um banco de dados relacionais. Pode ser usada de 3 maneiras: Pode ser chamado diretamente para transferir dados entre um documentos XML e o banco de dados. Pode ser chamado para transferir dados entre um documentos XML e o banco de dados, também pode ser usado como um mecanismo de ligação, neste caso, o usuário gera uma CLASSE específica para um dado em um documento XML ou tabelas de um banco de dados, então usa essa CLASSE diretamente em seu aplicativo.

5.2 SISTEMAS RELACIONAIS COM SUPORTE PARA XML

5.2.1 DB2

O DB2 da IBM oferece suporte ao XML através do DB2 XML Extender e DB2 Text Extender. O DB2 XML Extender permite o acesso e armazenamento a documentos XML, e também a criar documentos XML a partir de dados relacionais existentes ou converter documentos XML para dados relacionais. Com o DB2 XML Extender é possível utilizar o SQL como seu principal método de acesso aos documentos XML, facilitando também sua tarefa de administração através de um ambiente gráfico. O XML Extender gerencia documentos DTDs em suas próprias tabelas, sem a necessidade do outro aplicativo e com isso agiliza o processo. O XML Extender pode enviar documentos XML e recuperar documentos XML, validar documentos XML de acordo com "XML Schemas" ou DTDs, transformar documentos XML com o XSLT, copiar documentos XML de arquivos e banco de dados e extrair valores de documentos XML.

O DB2 Text Extender possui mecanismos de busca, como busca por sinônimo e busca

por sentença ou parágrafo. Pode ser usado quando um documento XML é armazenado em colunas simples e "XML-Aware" para expandir a busca dentro das seções de documentos estruturados.

5.2.2 Oracle 9i

O Oracle 9i XDB, da Oracle Inc., oferece suporte tanto à dados com suporte a XML ou a dados XML nativo, pode-se dizer que é o limite entre dados relacionais e dados XML. Possibilita o uso de comandos SQL que permite aos usuários ver dados relacionais como dados XML e dados XML como se fossem dados relacionais. A característica principal é o XMLType, é um tipo de objeto pré-definido que pode armazenar um documento XML. Como qualquer tipo de objeto, XMLType pode ser usado como um tipo de dados de uma coluna em uma tabela ou "view". Pode se parecer com um XML "view" (um documento XML virtual) pode ser construído sobre qualquer dado, não importando se são dados relacionais ou XML.

Um grande número de operações teve que ser adicionado ao SQL para que fosse possível ver dados XML como se fossem dados relacionais e vice-versa.

Outra grande vantagem que o XDB oferece é o repositório XML, que proporciona um arquivo de sistema como se fosse um objeto XMLType em um banco de dados, objetos XMLType podem ser associados como um caminho ou um URL correspondente no repositório hierárquico. O repositório mantém a propriedade de cada objeto, proprietário, data de modificação, versão e controle de acesso.

Usando o "Oracle XML Parser", você pode manipular documentos XML em seu "infoset", manipular seu conteúdo e retornar esse "infoset" alterado para o formato XML.

Usando o "Oracle XSLT Processor", você pode transformar XML em XML, HTML, ou texto com outra estrutura. Tanto o "Oracle XML Parser" com o "Oracle XSLT processor"

compartilham o “Oracle XPath Engine” que permite consultas ao infoset” de um documento XML para selecionar os “subsets” importantes dessa informação. O Oracle XML SQL Utility facilita a tarefa de criação de um XML a partir de um resultado de uma pesquisa SQL e o armazenando do documento XML em tabelas no banco de dados.

“Oracle Intermedia Text” habilita a indexação ou “queries” em documentos XML ou fragmentos do documento, mantendo a estrutura do XML em uma querie.

5.2.3 SQL Server 2000

Pode-se dizer que o SQL Server 2000, da Microsoft, tem suporte nativo a XML, pois pode importar e exportar dados para o banco sem a utilização de ferramentas externas.

O SQL Server é oferece integração com o XML, podendo importar e exportar dados neste formato e permitindo também uma integração com o servidor de Web da Microsoft, o Internet Information Server, para o envio de instruções SQL através de um navegador de internet e o retorno dos dados em XML.

Com o SQL Server 2000 o acesso um banco de dados relacional já existente através da XML é feito com o comando FOR XML, que retorna dados XML de um SELECT e de palavras chaves OPENXML T/SQL.

Outras características do produto relacionadas à XML são:

OpenXML, que permite uma visão relacional dos dados XML, com o T-SQL, é possível fazer um pedido de dados o XML, incluir dados XML em uma tabela relacional, e atualizar o banco de dados;

XML Updategrams, uma forma para atualizar, incluir, excluir dados um banco de dados SQL Server 2000 através do XML;

SQLXML, que habilita o suporte ao XML para o Banco de Dados SQL Server 2000, e permite ao desenvolvedor uma ligação entre o XML e os dados relacionais, podendo criar

“XML Views” dos dados relacionais existentes e trabalhar com eles, como se fosse um arquivo XML. O SQLXML é a solução para pesquisas no banco de dados relacional, pesquisas com SQL resultando em XML, atualizações de bancos de dados relacionais como se fossem XML, conversão de arquivos de XML para tabelas relacionais, e pesquisas através de URL a partir do navegador de internet, dentre outras características.

5.3 SISTEMAS NATIVOS EM XML

5.3.1 *Tamino XML Server*

O Tamino XML Server, da empresa Software AG, é um banco de dados com suporte nativo a XML. Ele é uma plataforma de armazenamento de informações XML sem a necessidade de conversão para outro formato, como tabelas relacionais. Integra dados relacionais e orientado a objetos em uma estrutura XML, podendo acessar banco de dados existentes remotos que são usados por aplicativos e pode armazenar esses dados como objetos XML. Pelo fato de armazenar documentos XML em um servidor banco de dados com suporte nativo à XML, ele consegue manter o sistema rodando com uma boa performance, escalabilidade e baixo custo de operação e administração.

O sistema XML utilizado no Tamino é multitarefa, permitindo um grande volume de dados e requisições de vários usuários simultâneos.

5.3.2 *Lore*

Lore desenvolvido primeiramente na universidade de Stanford é um banco de dados desenvolvido para armazenar dados semi-estruturados, foi modificado recentemente para um banco de dados XML. Inclui uma linguagem para "query" (Lorel), múltiplas técnicas de indexação, otimização "cost-based query", suporta multi-usuário, log e restauração, também

ode importar dados externos. Como foi desenvolvido para usar dados semi-estruturados, ele pode armazenar sem problema nenhum documentos XML sem DTDs.

Uma característica interessante do Lore é o "DataGuide" que é um sumário estrutural de todos os caminhos do BD, os dados são adicionados, então essa estrutura é sumarizada, esse tipo de informação é útil em um "query".

6. INTRODUÇÃO AO XINDICE

Neste capítulo é apresentado o modo básico de funcionamento do produto XIndex, que será o servidor de bancos de dados em nossa implementação de referência de bancos de dados nativos em XML. Inicialmente iremos apresentar os locais onde o produto pode ser baixado bem como seus passos de instalação. A seguir apresentamos seus principais comandos de operação e finalizando com um pequeno programa em Java que acessa e manipula dados de uma coleção do XIndex.

6.1 INTRODUÇÃO

O XIndex [<http://xml.apache.org/xindex>] é um banco de dados XML nativo. Isso significa que sua unidade de armazenamento é o próprio XML. O Xindex é um produto comercial produzido pela The Apache Software Foundation [<http://www.apache.org>], que faz parte do projeto Apache XML [<http://xml.apache.org>].

O XIndex possui uma implementação da API XML:DB [<http://www.xmldb.org>] para desenvolvimento com Java e utiliza a linguagem XPath como linguagem de consulta e a XUpdate como linguagem para atualização dos dados.

6.2 CARACTERÍSTICAS

Entre as principais características do XIndex, de acordo com informações disponíveis no site [<http://xml.apache.org/xindex>], podemos citar:

- Armazenamento dos documentos XML sob a forma de coleções, sendo que essas coleções podem ser de documentos com a mesma estrutura ou com estruturas diferentes, que podem ser consultados facilmente;

- XPath Query Engine, que utiliza a especificação XPath do W3C para consultar coleções, sendo um mecanismo bastante flexível para consulta;
- Indexação dos documentos XML para melhor performance nas consultas a grandes quantidades de dados;
- Implementação da linguagem XUpdate da XML:DB. A XUpdate consiste em uma linguagem com notação XML para atualização de dados em documentos XML;
- Implementação da API XML:DB para Java, facilitando o uso do XIndex como base de dados para programas feitos em Java;
- Implementação da XMLObjects, tecnologia que permite estender o servidor, agregando novas funcionalidades ao XIndex. O conceito da XMLObjects é bastante parecido com o de Stored Procedures (Procedimentos armazenados) existentes nos sistemas relacionais;
- Um conjunto de operações via linha de comando para administração do banco de dados;
- Compatibilidade com CORBA (Common Object Request Broker Architecture – tecnologia de componentes distribuídos desenvolvida pela Object Management Group - OMG), através da API “CORBA Network”.

Embora já possua muitas das principais características dos grandes sistemas de banco de dados, o próprio site possui uma grande lista de características que estão sendo buscadas para versões futuras, entre elas:

- Propriedades ACID;
- Melhora de performance;
- Melhor integração com outros produtos Apache;
- Agendamento de operações;
- Suporte a XQuery (linguagem de consulta XML do W3C).

6.3 COMO OBTER UMA DISTRIBUIÇÃO

O XIndex é um produto open-source (de código fonte aberto) e pode ser baixado gratuitamente no próprio site do produto. O endereço completo para baixar o programa é <http://xml.apache.org/xindice/download.html>. No momento em que fazíamos o trabalho, a versão estável para download era a 1.0. O XIndex é encontrado em um arquivo compactado em formato zip (plataforma win32) e em formato tar.gz (plataforma linux) e nossa implementação foi baseada na plataforma Microsoft Windows 2000.

6.4 REQUISITOS PARA INSTALAÇÃO

Não existem requisitos de hardware. Nossa instalação foi em um equipamento Intel Pentium II 333 com 128 megabytes de memória RAM e 2 gigabytes de espaço em disco, que apresentou resultado bastante satisfatório. O único requisito de software é que se tenha instalada uma cópia do software Development Kit da Sun (Sun SDK) versão 1.3 ou superior. Nesse trabalho utilizamos o Sun SDK 1.4.1_03, obtido a partir do endereço <http://java.sun.com/j2se/1.4.1/download.html>.

6.5 INSTALANDO O XINDEX

Uma vez com o arquivo baixado, descompacte-o em um diretório qualquer (por exemplo c:, pois o arquivo compactado já traz um diretório xml-xindice-1.0 embutido).

É necessária a criação/manutenção de quatro variáveis de ambiente para o correto funcionamento da aplicação. Em primeiro lugar, acrescente o diretório BIN do Xindex na variável PATH, para que o banco possa ser chamado de qualquer lugar, via linha de comando.

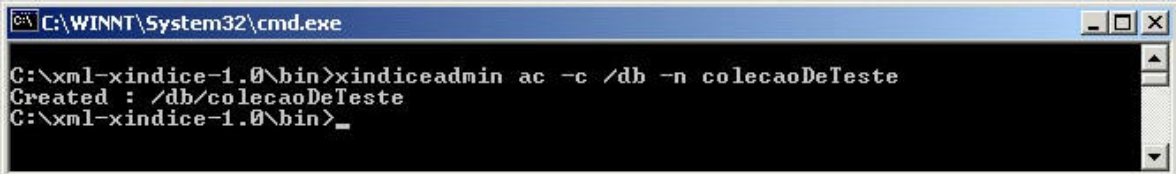
A seguir, crie uma variável JAVA_HOME, que deverá apontar para o diretório onde o SDK da Sun está instalado. Posteriormente, crie uma variável chamada XINDICE_HOME, apontando para o diretório onde o Xindice foi instalado. Finalmente, acrescente o arquivo XINDICE.JAR, encontrado no diretório Java/lib do seu Xindice na variável CLASSPATH.

Seguindo os passos acima, seu servidor de banco de dados estará pronto para uso. Para testa-lo, acesse o endereço “http://localhost:4080/” pelo seu navegador e veja a página web do Xindice, caso a instalação tenha sido concluída com sucesso.

6.6 COMANDOS DE OPERAÇÕES

O XIndice oferece uma série de operações via linha de comando para administração e manutenção do sistema de banco de dados. A seguir mostraremos as principais operações do XIndice.

6.6.1 Criando uma coleção



```
C:\WINNT\System32\cmd.exe
C:\xml-xindice-1.0\bin>xindiceadmin ac -c /db -n colecaoDeTeste
Created : /db/colecaoDeTeste
C:\xml-xindice-1.0\bin>_
```

Figura 1: Comando para criar uma coleção no XIndice

O comando cria uma coleção chamada colecaoDeTeste.

Vamos analisar cada parâmetro passado a este comando. O primeiro parâmetro “ac” vem das iniciais de add_collection, e significa que estamos adicionando uma coleção.

O segundo parâmetro “-c” indica que iremos entrar com o contexto onde o comando deve ser executado. Passamos, então, a raiz do XIndice como contexto.

O terceiro parâmetro indica que a coleção será criada dentro da raiz do XIndice,

caracterizada como “/db”.

O quarto parâmetro “-n” indica que iremos entrar com o nome da coleção. No caso passamos o nome “colecacaoDeTeste”.

Ao executar o comando, o XIndice retorna o status do comando recém executado. Se esse for completado com sucesso, a frase “Created db/nomeDaColecacao” será retornada. Caso a coleção já exista, um erro é retornado “ERROR: Collection Duplicated”.

Utilizando a mesma estrutura de comandos, poderíamos ainda listar e excluir uma coleção, bastando para isso utilizar os parâmetros lc e dc, respectivamente, no lugar de ac.

6.6.2 Adicionando um documento a uma coleção

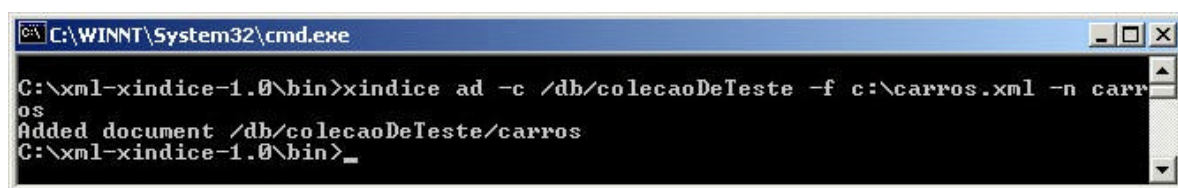


Figura 2: Comando para adicionar um documento a uma coleção

O comando adiciona o arquivo carros.xml, localizado em c:\ à coleção chamada “colecacaoDeTeste”. Para isso estamos assumindo que existe um arquivo chamado carros.xml localizado em c:\ com a seguinte estrutura:

```
<?xml version="1.0"?>
<carros>
  <carro chassis="1111111">
    <placa>ABC-1234</placa>
    <cor>Vermelho</cor>
    <marca>Volkswagen</marca>
    <modelo>Gol</modelo>
    <ano>2002</ano>
  </carro>
  <carro chassis="2222222">
    <placa>DEF-1234</placa>
    <cor>Branco</cor>
    <marca>Ford</marca>
    <modelo>Fiesta</modelo>
  </carro>
</carros>
```

```

        <ano>2003</ano>
    </carro>
</carros>

```

Arquivo carros.xml, utilizado para exemplificar o comando para adicionar um documento a uma coleção no XÍndice.

6.6.3 Recuperando dados

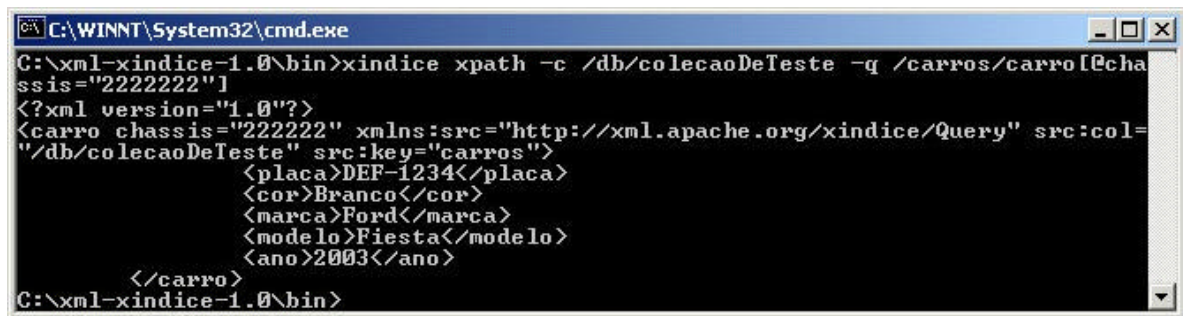


Figura 3: Exemplo de consulta a uma coleção, utilizando XPath.

As operações de consulta às coleções no XÍndice fazem uso da linguagem XPath. Sua sintaxe básica consiste em especificarmos o contexto onde a busca será feita, através do parâmetro “-c” e a query XPath através do parâmetro “-q”.

6.6.4 Atualizando registros

A atualização de dados no XÍndice é feita utilizando-se a linguagem XUpdate. Para isso é necessário o desenvolvimento de um pequeno programa em Java para executar as funções de atualização.

Um exemplo de instrução XUpdate é o XUPDATE:INSERT-BEFORE, que inclui um novo nó ao documento imediatamente antes de um nó especificado. Sua sintaxe é bastante simples, como segue:

```

<xupdate modifications version="1.0" xmlns:xupdate=http://xmldb.org/xupdate>
  <xupdate:insert-before select="/carros/carro[@chassis='1111111']/placa">

```

```

        <xupdate element name="cilindradas">1800</xupdate:element>
    </xupdate:insert-before>
</xupdate:modifications>

```

Exemplo 1: Exemplo de instrução XUpdate

O XUpdate fornece ainda várias outras operações que não cabem nesse trabalho. Para maiores informações, consulte diretamente o site do projeto XUpdate, em [\[http://www.xmldb.org/xupdate\]](http://www.xmldb.org/xupdate), onde encontrará toda a especificação do XUpdate.

Segue abaixo listagem de um programa em Java que executa a instrução acima.

```

import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.xmldb.api.*;

public class CarrosXUpdate {

    public static void main(String[] parametros) throws Exception {

        Collection col = null;
        try {
            String driver = "org.apache.xindice.client.xmldb.DatabaseImpl";
            Class c = Class.forName(driver);
            Database database = (Database) c.newInstance();
            DatabaseManager.registerDatabase(database);
            col= DatabaseManager.getCollection("xmldb:xindice:///db/colecaoDeTeste");
            String xup = "";
            xup+="

```

Exemplo 2: Programa escrito em Java que executa uma instrução XUpdate em uma coleção do XIndice

As três primeiras linhas do programa destinam-se a importar todas as classes da API do XÍndice, que iremos utilizar para conectar-se ao banco e utilizar seus serviços.

Criamos, então, a classe principal do programa e a seguir iremos definir nosso método principal (main). Nele criamos um objeto do tipo Collection, que irá armazenar uma réplica da coleção, para manipulação.

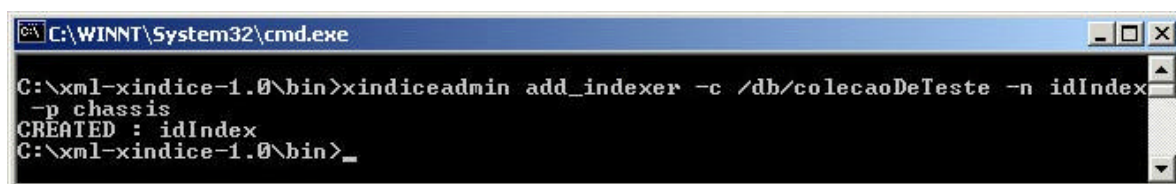
Criamos então uma variável do tipo texto para armazenar o nome do driver que utilizaremos para acesso ao banco. Instanciamos dinamicamente um objeto para o driver, através da instrução `Class.forName(driver)`.

Uma vez com o driver carregado, vamos criar um objeto do tipo Database, que será posteriormente registrado no DatabaseManager.

Populamos nosso objeto do tipo Collection através do método `getCollection` do DatabaseManager. Nesse ponto passamos uma string indicando qual coleção queremos trazer para a memória, no nosso caso, `colecacaoDeTeste`.

Criamos outra variável do tipo texto contendo a cláusula `XUpdate` que queremos aplicar à coleção. Criamos a seguir um objeto que nos fornecerá o serviço para o `XUpdate` e passaremos a este objeto nossa cláusula. Assim ao término do processamento teremos nossa atualização efetuada.

6.6.5 Criando Índices



```
C:\WINNT\System32\cmd.exe
C:\xml-xindice-1.0\bin>xindexadmin add_indexer -c /db/colecacaoDeTeste -n idIndex
-p chassis
CREATED : idIndex
C:\xml-xindice-1.0\bin>_
```

Figura 4: Comando para criar um índice em uma coleção

O XÍndice trabalha com índices para melhorar a performance das consultas. O

comando acima cria um índice para o atributo chassis na coleção colecaoDeTeste.

6.7 CONSIDERAÇÕES

Por estes exemplos simples, pôde-se verificar que o XÍndice possui uma linguagem de consulta e atualização bastante robusta e de fácil entendimento, além de um bom esquema de indexação, melhorando bastante a performance do banco. O XÍndice possui muitas características que o colocam como alternativa para armazenamento de sistemas de pequeno porte, bem como para sistemas em dispositivos de pequeno porte, como handhelds e laptops.

No próximo capítulo apresentaremos uma aplicação baseada no XÍndice, com o objetivo de conferir as vantagens e desvantagens do desenvolvimento com base em um banco de dados XML nativo.

7. IMPLEMENTAÇÃO

Nesse capítulo estaremos mostrando e discutindo a implementação de um pequeno sistema baseado no banco de dados XML Nativo XIndex. Essa implementação foi feita para ilustrar o uso de um banco de dados XML Nativo, bem como avaliar os resultados, visando obter as reais vantagens de se utilizar esse tipo de banco de dados.

7.1 APRESENTAÇÃO / MODELAGEM DO SISTEMA

O sistema consiste em um controle de contatos, que guarda informações das ligações feitas para clientes e/ou fornecedores, montando uma base de conhecimento para posterior consulta.

As entidades do sistema são representadas pelo diagrama abaixo:

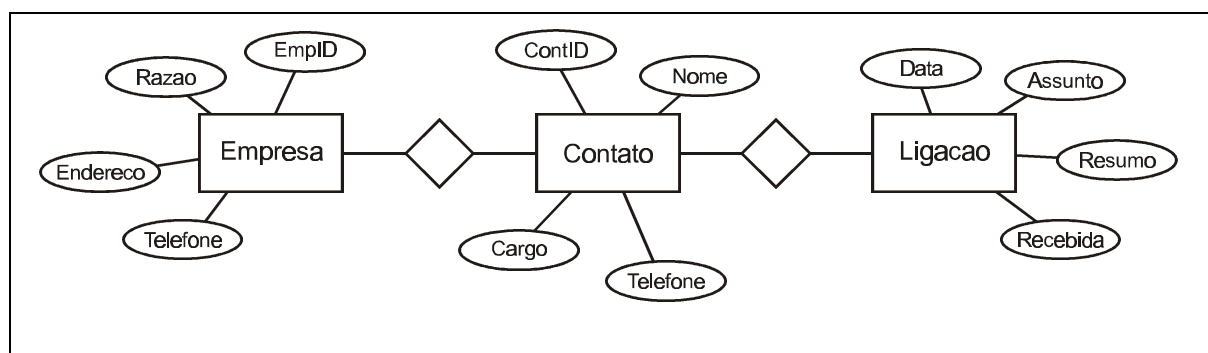


Figura 1: Diagrama Entidade-Relacionamento das entidades do sistema

As entidades acima foram modeladas para que pudessem ser representadas em um modelo XML. Nesse ponto poderíamos ter optado por várias abordagens diferentes. Uma primeira abordagem seria utilizar um esquema relacional, onde teríamos documentos XML distintos com algum atributo servindo como elo de ligação (agindo como uma chave estrangeira). Esse esquema é chamado Modelo de Dados XML orientado a Relacionamentos,

e poderia ser representado por:

```
<?xml version="1.0"?>
  <Empresas>
    <empresa codEmpresa="1">
      <atributos>...</atributos>
    </empresa>
    ...
  </Empresas>
```

Exemplo 1: Documento de empresas

```
<?xml version="1.0"?>
  <contatos>
    <contato codEmpresa="1" codContato="1">
      <atributos>...</atributos>
    </contato>
    ...
  </contatos>
```

Exemplo 2: Documento de contatos

```
<?xml version="1.0"?>
  <ligacoes>
    <ligacao codEmpresa="1" codContato="1">
      <atributos>...</atributos>
    </ligacao>
    ...
  </ligacoes>
```

Exemplo 3: Documento de ligações

Para nosso sistema, escolhemos uma outra abordagem, que consiste em um modelo mais próximo da XML, para não cairmos em uma simples representação de um modelo relacional em XML. Escolhemos uma modelagem chamada de Modelo de Dados XML orientado a nós. Como podemos perceber, um elemento na XML pode possuir outros elementos filhos, que por sua vez possuem um único elemento pai. Aproveitando-se dessa característica, podemos representar nosso sistema de duas maneiras: utilizando-se de um único documento ou de documentos separados para cada empresa.

Na primeira, a representação ficaria parecida com o documento abaixo:

```

<?xml version="1.0"?>
<Empresas>
  <empresa empID="1">
    <atributos>...</atributos>
    <contatos>
      <contato contID="1">
        <atributos>...</atributos>
        <ligacoes>
          <ligacao>
            <atributos>...</atributos>
          </ligacao>
        </ligacoes>
      </contato>
    </contatos>
  </empresa>
  ...
</Empresas>

```

Exemplo 4: Documento XML para o sistema de contatos com modelagem orientada a nós ("nodes"), com múltiplas empresas em um único documento.

Esse modelo funciona perfeitamente para representar os dados de que precisamos, porém não é a melhor escolha. Um banco de dados XML Nativo é organizado para trabalhar com coleções de documentos XML, e não para um único documento XML que contém todos os dados. Ao utilizar um esquema como o ilustrado acima, teríamos um único documento XML de tamanho considerável, cujo custo de acesso e consulta cresceria exponencialmente à medida em que fossem incluídos novos registros.

Um modelo parecido foi adotado, porém fragmentado em vários documentos menores, um para cada empresa. O modelo final que foi utilizado é o apresentado abaixo:

```

<?xml version="1.0"?>
<empresa empID="1">
  <atributos>...</atributos>
  <contatos>
    <contato contID="1">
      <atributos>...</atributos>
      <ligacoes>
        <ligacao>
          <atributos>...</atributos>
        </ligacao>
      </ligacoes>
    </contato>
  </contatos>
</empresa>

```

Exemplo 5: Documento XML com modelagem orientada a nós (nodes) e uma empresa por documento.

7.2 PLATAFORMA

O sistema foi desenvolvido em linguagem Java, sendo criado como uma aplicação Web. Todo o sistema foi desenvolvido tendo uma classe Java normal, responsável pela conexão ao banco de dados, cinco classes representando as entidades do sistema e várias classes responsáveis pelos Servlets e Java Server Pages (Servlets e Java Server Pages são tecnologias Java utilizadas na geração de páginas web dinâmicas desenvolvidas pela Sun Microsystems), além das APIs do XIndice e algumas outras de manipulação de XML.

A aplicação foi desenvolvida, instalada utilizando o sistema operacional Microsoft Windows 2000 e servidor de aplicações Apache Tomcat, versão 4.1 [<http://jakarta.apache.org>].

Para o desenvolvimento, utilizamos o Eclipse [<http://www.eclipse.org>], ambiente integrado de desenvolvimento feito em Java, desenvolvido pela IBM.

7.3 ARQUITETURA DO SISTEMA

O sistema consiste em controlar cadastro e manutenção de empresas, contatos e ligações. As ações que podem ser feitas pelo usuário são definidas no diagrama de Use Cases abaixo e serão detalhadas na tabela 1.

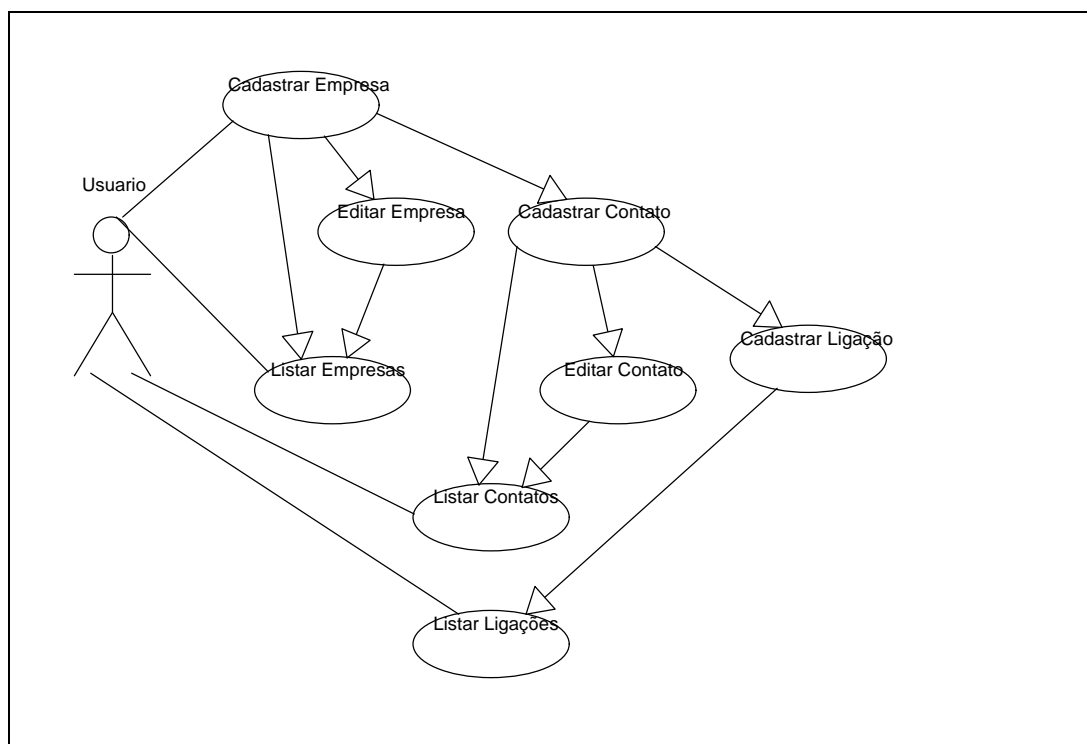


Figura 2: Diagrama de Use Cases para o sistema de contatos

Ação	Descrição
Cadastrar Empresa	Ato de entrar com os dados de uma nova empresa no sistema. O usuário irá entrar com os dados da empresa, preparando o sistema para a posterior entrada de contatos e ligações.
Editar Empresa	Ato de alterar os dados de pré-cadastrados de uma empresa.
Listar Empresa	Ato no qual pode-se buscar dados de uma empresa.
Cadastrar Contato	Após a inclusão das empresas, pode-se entrar com os dados de um novo contato para uma determinada empresa. Essa ação dará suporte para que sejam incluídos os dados de ligações.
Editar Contato	Ato de alterar dados pré-cadastrados de contatos.
Listar Contato	Ato no qual pode-se buscar dados de um contato.
Cadastrar Ligação	Principal finalidade do sistema. Aqui serão entradas as informações relativas às ligações feitas ou recebidas a um determinado contato.
Listar Ligação	Ato no qual pode-se buscar os dados de uma ligação.

Tabela 1: Tabela de ações para o sistema de contatos

Optamos por desenvolver uma aplicação para a Web para utilizar nosso banco de dados. Criamos uma aplicação baseada em JSP + Servlets, utilizando um padrão de design de aplicação (Design Pattern) conhecido como Model View Controller (MVC) para desenvolvimento em três camadas, separando a camada de apresentação da camada de regras de negócios e da camada de dados.

O padrão MVC consiste em termos um único componente (Controller) que recebe

todas as solicitações da camada de apresentação (View). O controller passa as informações necessária à camada de regras de negócios e de dados (Model). Essa por sua vez processa as informações e as retorna para o Controller, que encaminha a resposta para a View.

Para suportar tal aplicação, desenvolvemos classes representadas na figura abaixo.

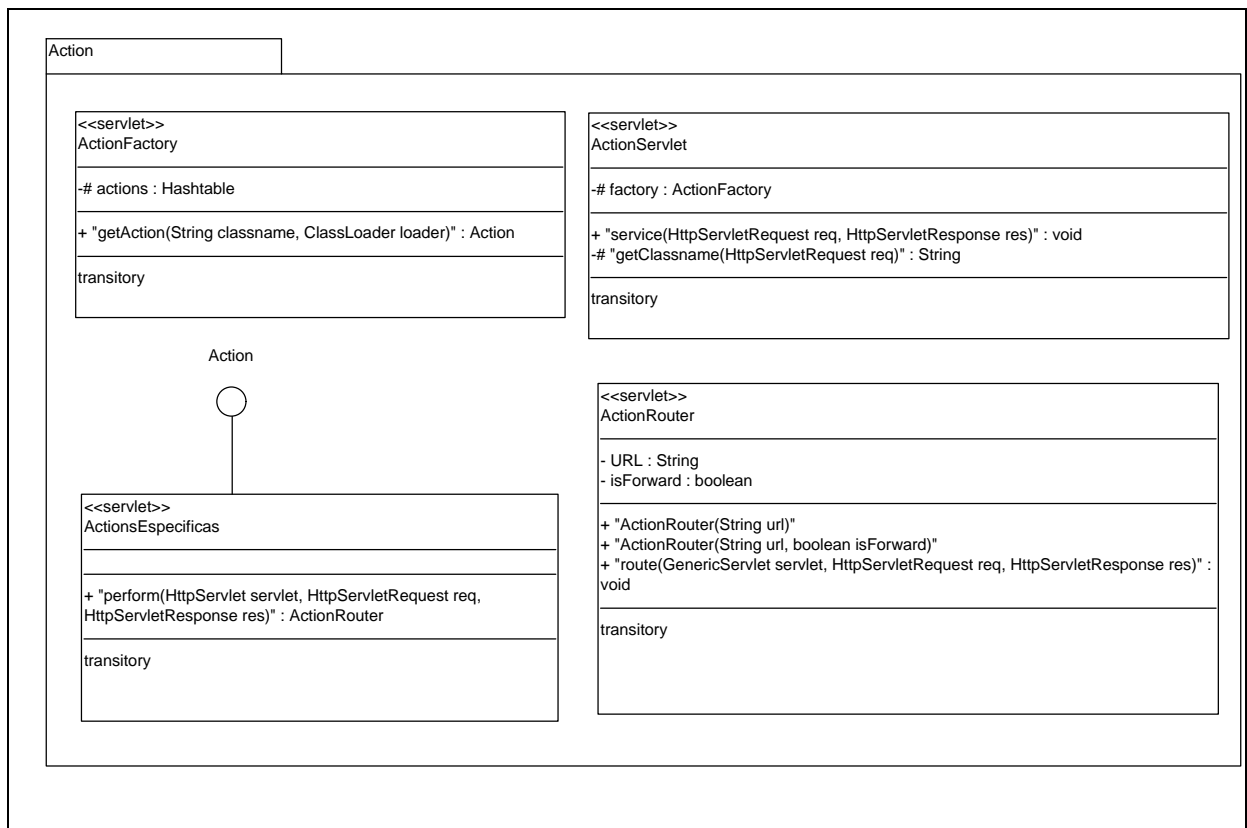


Figura 3: Diagrama de classes dos servlets responsáveis pelo modelo MVC da aplicação

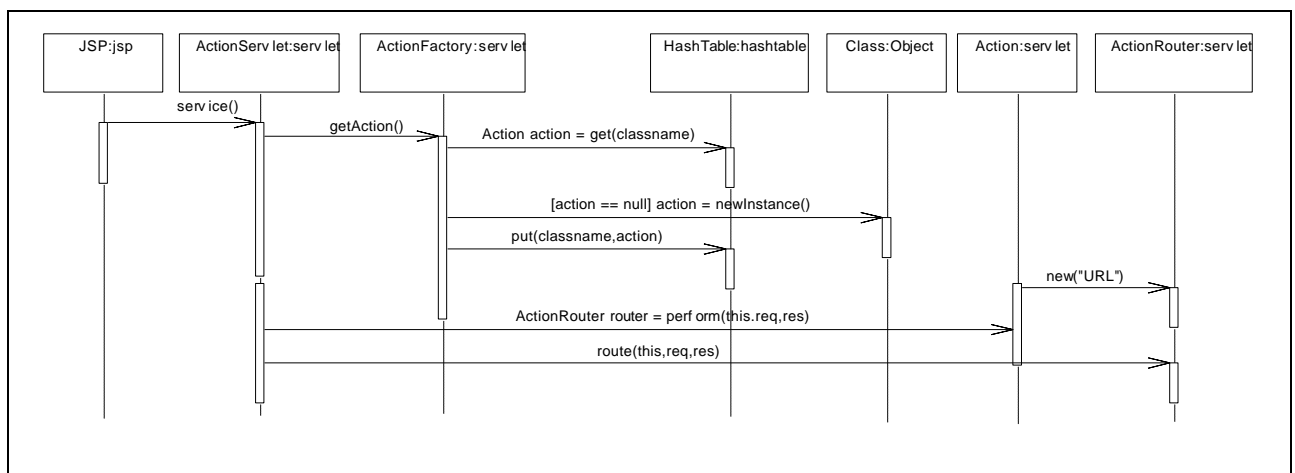


Figura 4: Diagrama de sequência das servlets responsáveis pelo modelo MVC da aplicação

Assim, todas as possíveis ações do sistema, como cadastrar empresa, editar contato ou listar ligações, por exemplo, são classes específicas que irão implementar a interface Action. Utilizando essa metodologia, fica simples implementar novas funcionalidades, sendo que para isso baste que essa nova funcionalidade seja uma classe que implemente a interface Action. Maiores detalhes sobre Design Patterns para aplicações Java fogem do escopo desse trabalho e podem ser encontradas em livros específicos, como Core J2EE Patterns [DEEPAK, 2002].

Utilizaremos um mapeamento de ações para nosso servidor de aplicações, mapeando todas as URLs que terminam com a extensão “.do” para o servlet controlador. A terminação “do” foi escolhida por ser fácil de se utilizar e de se entender (do é fazer em inglês). Assim, podemos acessar o servlet de cadastro de empresas como “addEmpresas.do”. Segue abaixo fragmento do arquivo XML utilizado para o mapeamento.

```
<web-app>
<display-name>TGI</display-name>
  <description>Trabalho de graduação interdisciplinar. Flávio
    Jimenez, Hilton Higasi, Lincoln Masao Muraoka Junior</description>
  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>ActionServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

Exemplo 6: Fragmento do arquivo web.xml, utilizado pelo Tomcat para configurar o servidor.

Para suportar todas as operações com o banco de dados XÍndice, criamos uma classe chamada DBConnection que é a responsável por gerenciar as conexões com o banco de dados. Ela implementa um Design Pattern conhecido como Singleton, que consiste em termos um construtor privado e que todas as classes que precisem utilizar serviços dessa classe utilizam uma instância comum, através de um método do tipo getInstance(). Abaixo segue o código da classe DBConnection (o código fonte completo encontra-se disponível na

Universidade Presbiteriana Mackenzie, para consulta).

```
package Core;

import org.xmldb.api.base.*;
import org.xmldb.api.DatabaseManager;
import javax.swing.*.*;

public class DBConnection{

    public String URL = "xmldb:xindice:///db/sistContatos/";
    protected static DBConnection connection = new DBConnection();
    protected static Database db = null;
    protected static Collection collection = null;
    public Collection getCollectionInstance() throws java.io.IOException{
        try{
            if(db == null){
                String driver;
                driver = "org.apache.xindice.client.xmldb.DatabaseImpl";
                db = (Database)
                    Class.forName(driver).newInstance();
                DatabaseManager.registerDatabase(db);
                collection = DatabaseManager.getCollection(URL);
            }
        }
        catch(Exception e){ e.printStackTrace(); }
        return collection;
    }
}
```

Exemplo 7: Código fonte da classe DBConnection, responsável pela conexão ao banco de dados

Além da classe de conexão, que está dentro de um pacote chamado Core, desenvolvemos também classes representando entidades, que serão enviadas para as páginas JSP. Essas classes foram colocadas no pacote chamado Beans (as classes auxiliares utilizadas por páginas JSP, e que geralmente representam entidades, são chamadas de Beans ou JavaBeans) e representam as entidades Empresa, Contato, Ligação. Essas classes possuem métodos getters e setters (de Get e Set do inglês, que significam, respectivamente obter e

atribuir) responsáveis pelo acesso aos seus atributos.

Finalmente, temos as classes que são responsáveis pelas regras de negócio do sistema. Como foi dito, todas as regras foram definidas como ações, que implementam a interface Action. Assim, foram criadas diversas classes para tarefas específicas, como addEmpresa.class, editEmpresa.class, searchEmpresa.class, etc. Essas classes possuem uma instância de DBConnection, para acesso ao banco de dados e possuem um método “perform”, onde será feito o processamento e o redirecionamento para a camada de apresentação.

Segue abaixo a classe addEmpresa, para ilustrar como foram implementadas as classes de ação:

```
package Action;

import javax.servlet.*;
import javax.servlet.http.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import Core.*;
import Beans.*;

public class addEmpresa implements Action{

    private DBConnection connection = new DBConnection();

    public ActionRouter perform(HttpServletRequest servlet,
                                HttpServletRequest req,
                                HttpServletResponse res)
                                throws java.io.IOException, ServletException{

        Collection col = null;
        String xmlDoc = "";
        xmlDoc += "<Empresa>";
        xmlDoc += "<empID>" + req.getParameter("codInt") + "</empID>";
        xmlDoc += "<razao>" + req.getParameter("razao") + "</razao>";
        xmlDoc += "<endereco>" + req.getParameter("endereco") + "</endereco>";
```

```

xmlDoc += "<telefone>" + req.getParameter("telefone") + "</telefone>";
xmlDoc += "</Empresa>";
try{

    col = connection.getCollectionInstance();
    XMLResource resource = _
        (XMLResource) col.createResource("", "XMLResource");
    resource.setContent(xmlDoc);
    col.storeResource(resource);
    sucessBean mensagem;
    mensagem = new sucessBean("Empresa cadastrada com sucesso.");
    req.getSession().setAttribute("mensagem",mensagem);
    return new ActionRouter("/sucesso.jsp");

}catch(XMLDBException e){
    errorBean erro;
    erro = new errorBean("Erro do XIndice");
    req.getSession().setAttribute("erro",erro);
    return new ActionRouter("/erro.jsp");

}catch(Exception e){
    errorBean erro2;
    erro2 = new errorBean("Exceção");
    req.getSession().setAttribute("erro",erro2);
    return new ActionRouter("/erro.jsp");

}
}
}

```

Exemplo 8: Código fonte da classe responsável pelo cadastro de novas empresas

O código fonte completo da aplicação pode ser obtido junto à Faculdade de Computação e Informática da Universidade Presbiteriana Mackenzie.

7.4 O SISTEMA EM AÇÃO

O sistema possui uma tela inicial com menus referentes a empresas, contatos e

ligações. Cada um desses itens de menu possui sub-itens relativos a cadastro, consulta e edição (menos no caso de ligações) dos dados.

A primeira ação possível é o cadastro de empresas, conforme a tela abaixo:

The screenshot shows a Microsoft Internet Explorer window titled 'Implementação - Controle de contatos'. The address bar displays 'http://lmmj:8080/TGI/root.jsp'. The page features a teal header with three tabs: 'Empresas', 'Contatos', and 'Ligações'. The 'Empresas' tab is active, revealing a form titled 'Cadastro de Empresas'. The form contains four input fields: 'Código da Empresa', 'Razão Social', 'Endereço', and 'Telefone'. A 'Cadastrar' button is positioned at the bottom right of the form. The browser's status bar at the bottom shows 'Concluído' and 'Intranet local'.

Cadastro de Empresas	
Código da Empresa	<input type="text"/>
Razão Social	<input type="text"/>
Endereço	<input type="text"/>
Telefone	<input type="text"/>
<input type="button" value="Cadastrar"/>	

Figura 5: Tela de cadastro de empresas

A seguir, pode-se editar ou excluir uma empresa, como segue:

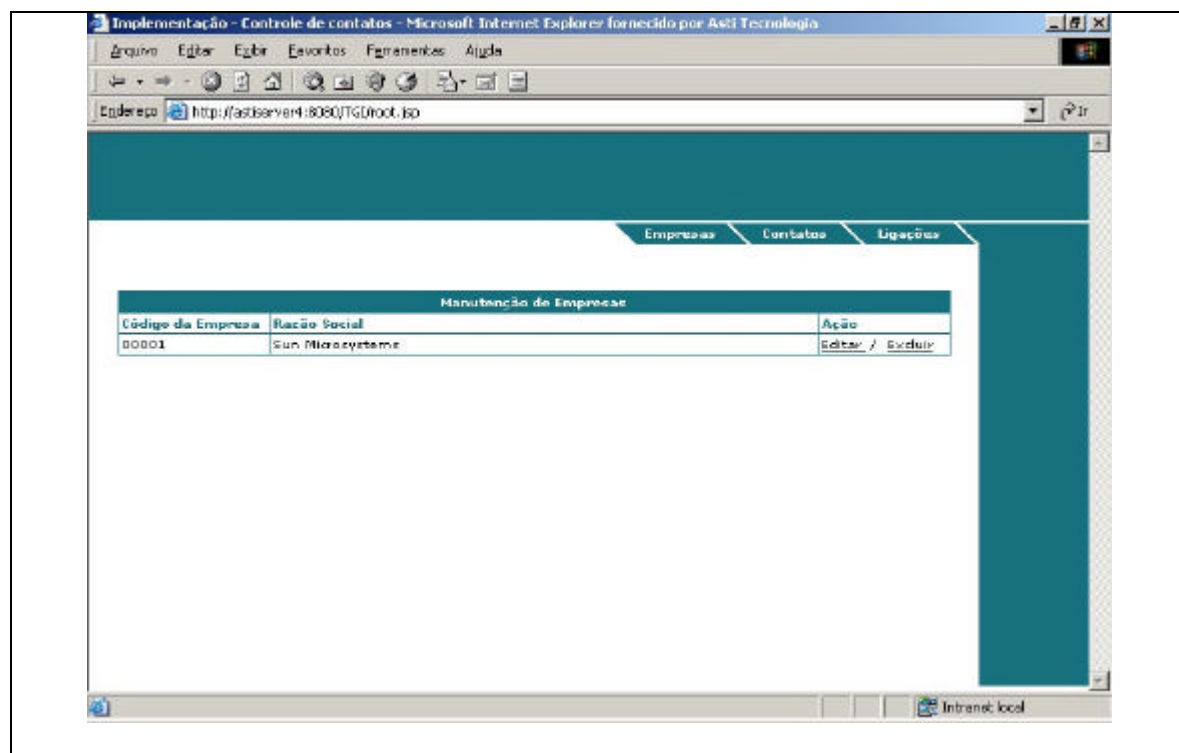


Figura 6: Tela de manutenção de empresas

Uma vez escolhida uma empresa, caso a opção seja editar, uma tela individual para edição será aberta:

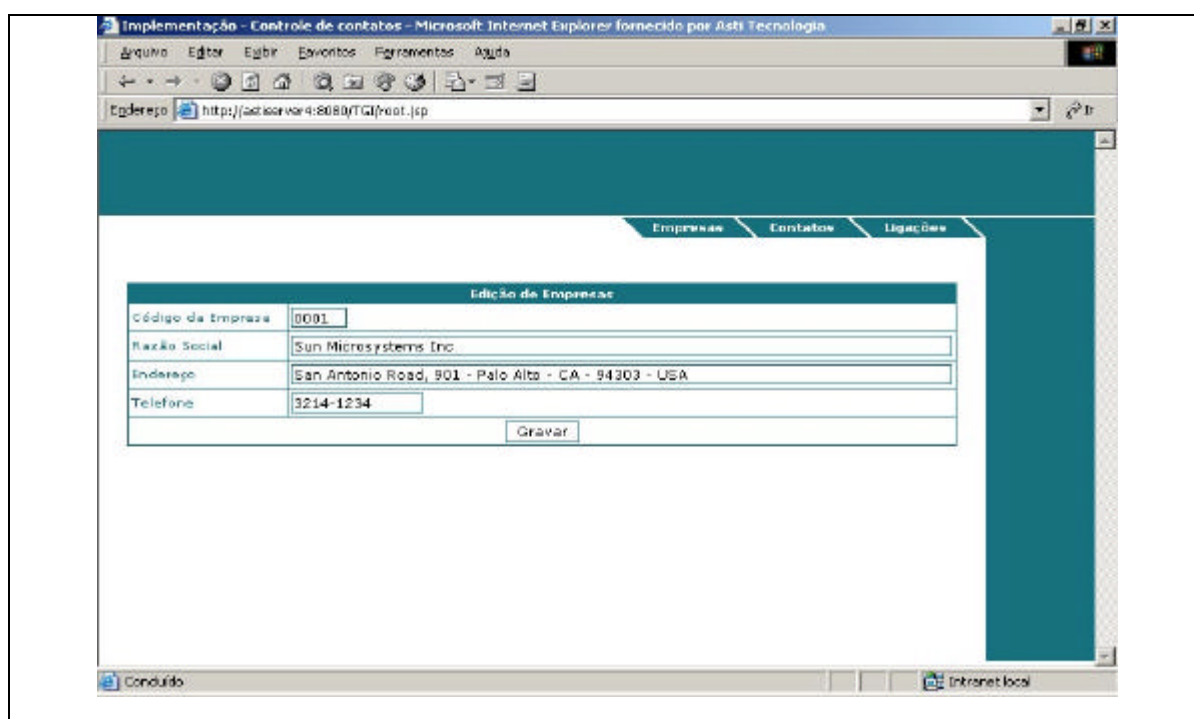


Figura 6: Tela de edição individual de empresas

Podemos ainda buscar uma empresa específica, como na tela apresentada abaixo:

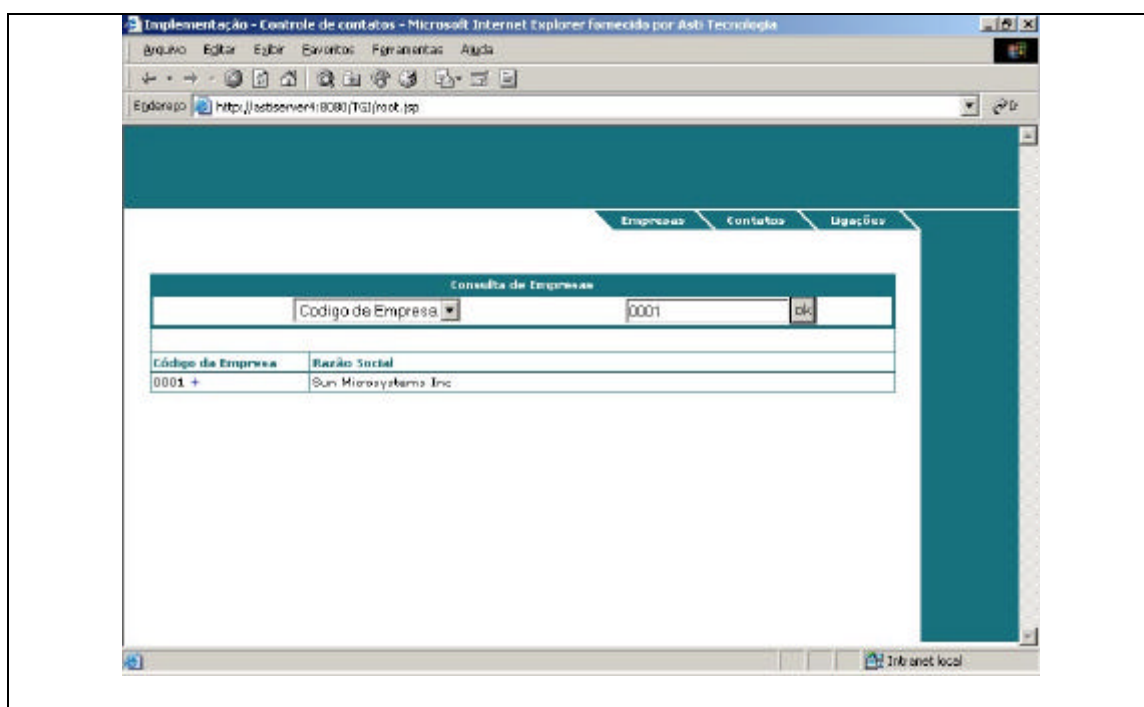


Figura 7: Tela de consulta de empresas

Uma vez cadastradas as empresas, podemos cadastrar os contatos para cada uma das empresas, conforme a tela abaixo:

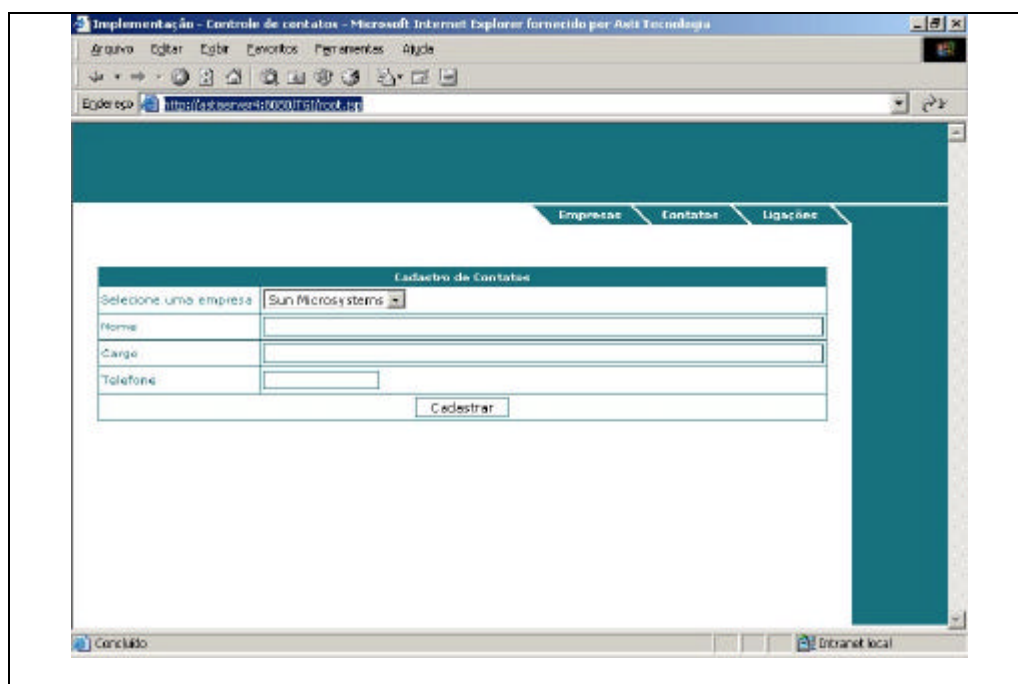


Figura 8: Tela de cadastro de contatos

Os contatos também podem ser editados, e a edição é feita em três passos. Primeiro escolhe-se uma empresa e todos os contatos da empresa são mostrados. A seguir escolhe-se um contato a ser editado ou excluído, conforme as telas seguintes:

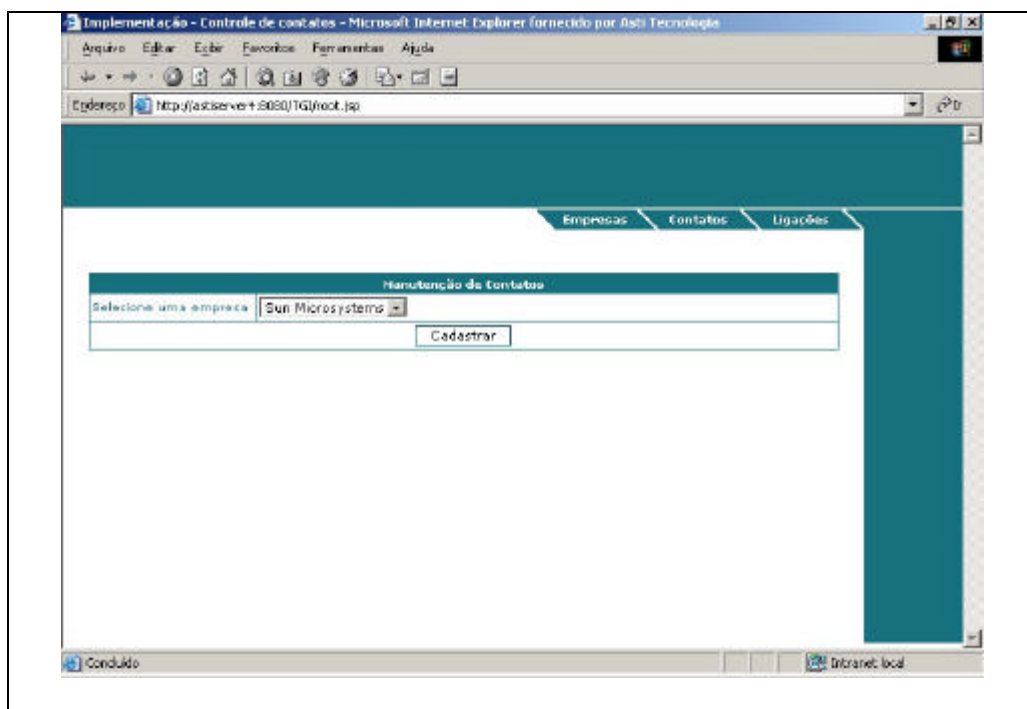


Figura 9: Tela de escolha da empresa

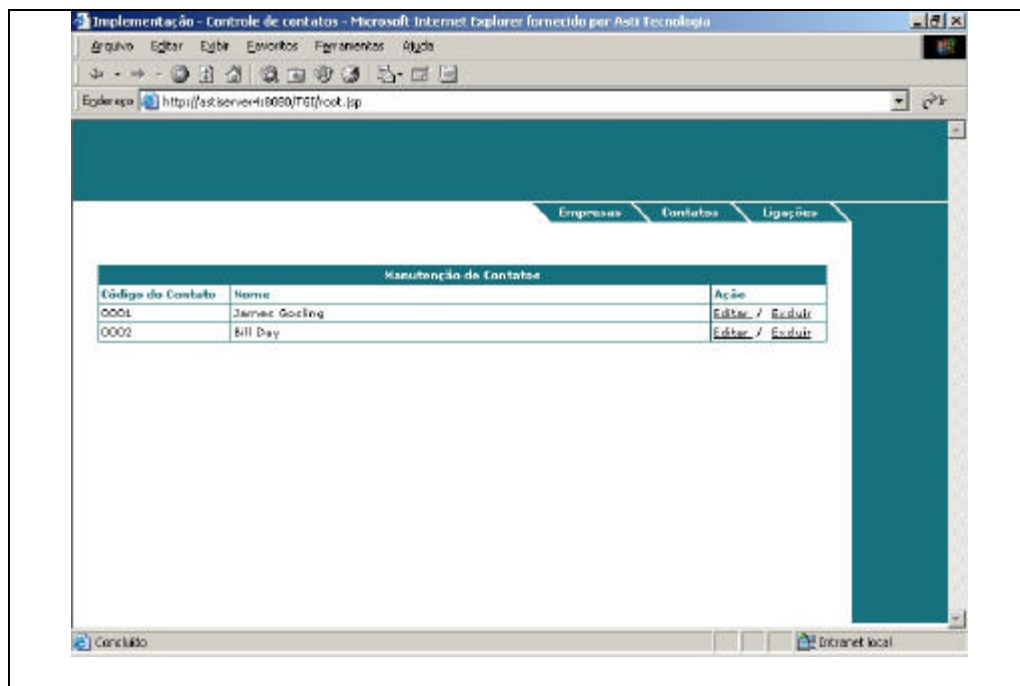


Figura 10: Telas de escolha de contatos

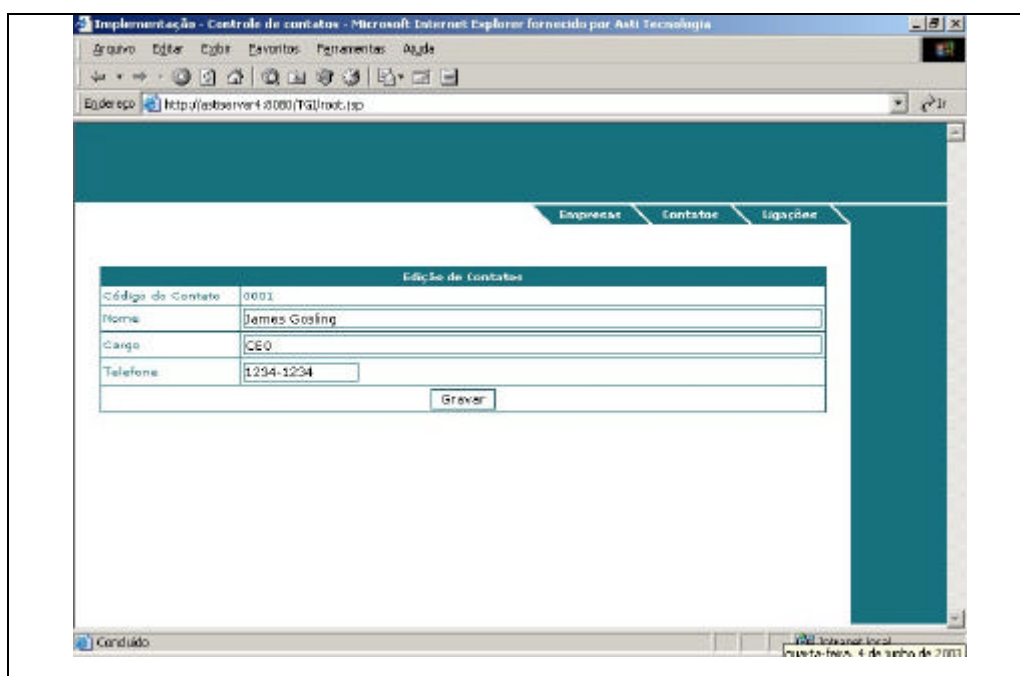


Figura 11: Tela de edição de contato

Assim como com os dados das empresas, é possível também realizar consultas nos dados de contatos. Essa pesquisa é feita conforme a tela abaixo:

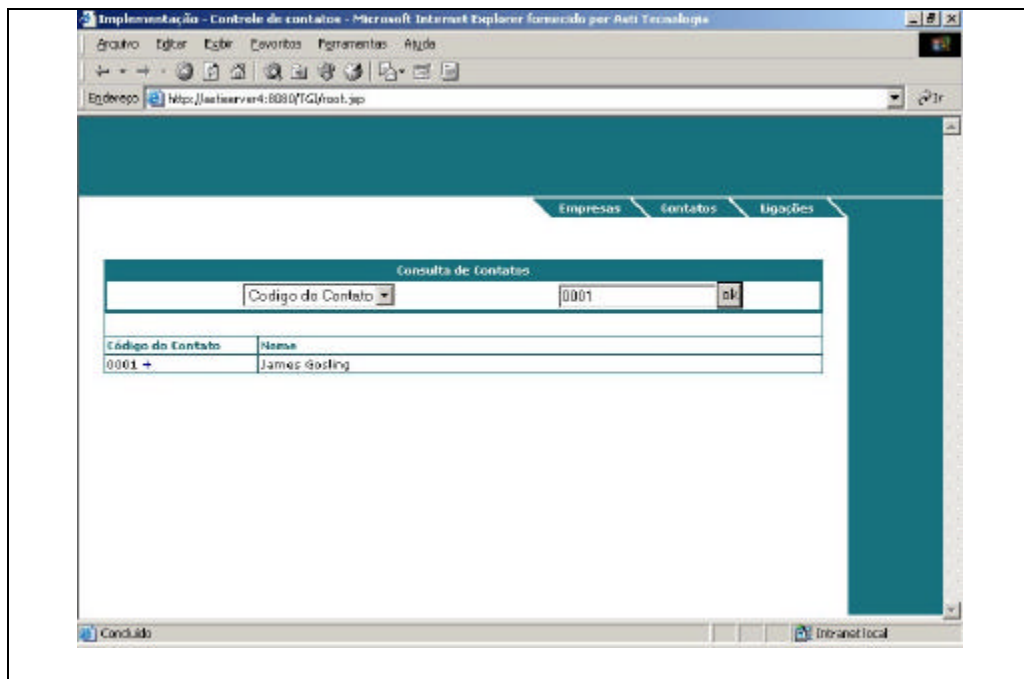


Figura 12: Tela de consulta de contatos

As últimas ações possíveis são as relativas às ligações. Para o cadastro de ligações, primeiro precisamos escolher uma empresa, em seguida um contato e preenche-se os dados da ligação:

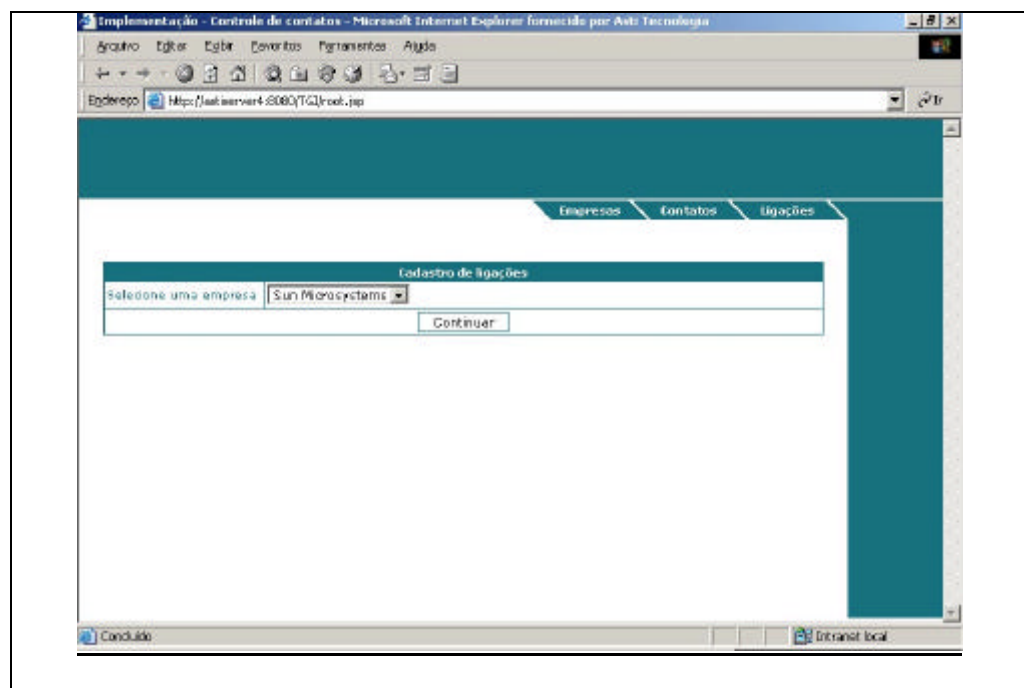


Figura 13: Tela de escolha de empresa

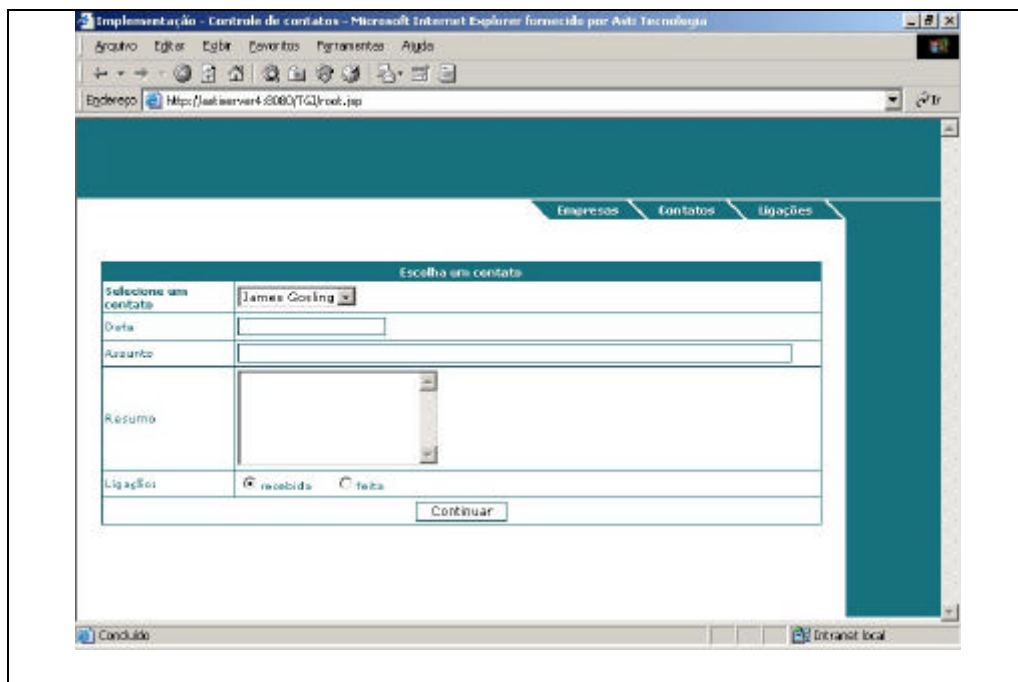


Figura 14: Tela de cadastro de ligação

Finalizando, a última ação possível é a busca de ligações, que pode ser feita utilizando a tela abaixo:

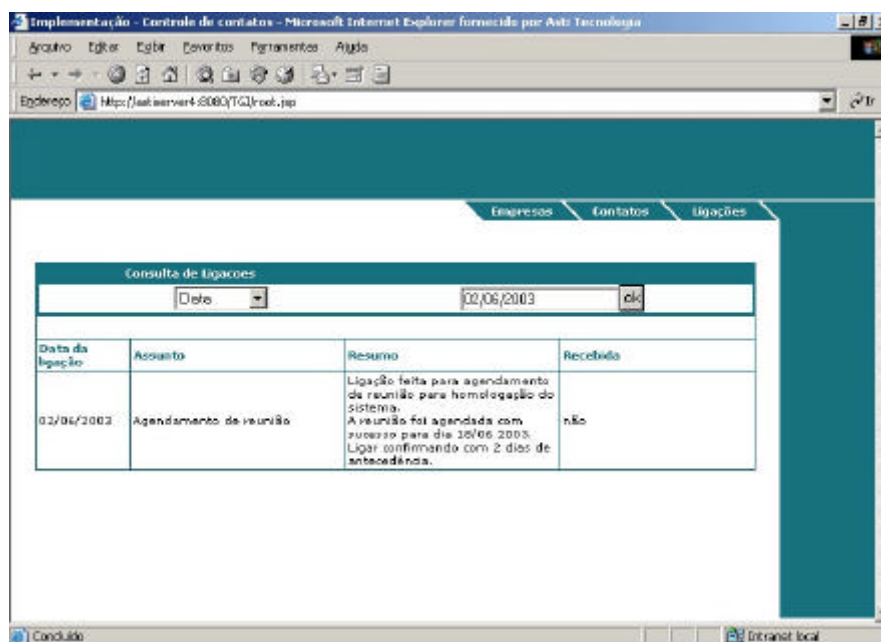


Figura 15: Tela de consulta de ligações

7.5 RESULTADOS

O desenvolvimento do sistema se deu de forma rápida e fácil, sendo que encontramos bastante documentação dos produtos utilizados.

Acreditamos que o resultado da implementação foi bastante satisfatório, pois o desempenho, que seria um dos determinantes pela escolha de um banco de dados XML Nativo foi bastante razoável, desde que o sistema a ser implementado seja de porte pequeno a médio.

Os componentes do grupo têm experiência em desenvolvimento de sistemas em outras plataformas de bancos de dados, e pudemos concluir que sua performance é compatível aos grandes bancos de dados, se forem cumpridas as condições estabelecidas acima.

8. CONCLUSÃO

As principais abordagens para se atingir índices mais altos de desempenho, confiabilidade e disponibilidade no armazenamento e recuperação de dados, representados por qualquer que seja o modelo lógico, sem dúvida continuam sendo os sistemas gerenciadores de bancos de dados.

Historicamente, o modelo relacional de representação dos dados assumiu um papel de suma importância nas aplicações corporativas, e ocupa até hoje um lugar de destaque no cenário dos bancos de dados. Mais recentemente, o surgimento e fortalecimento da XML como um padrão genérico e amplamente aceito para representação de estruturas de dados bem como para intercâmbio de informações gerou um novo desafio: manter o desempenho e confiabilidade do armazenamento e recuperação de informações que estejam na forma de XML.

A XML permite que representemos outras estruturas menos convencionais de dados, além das já conhecidas tabelas do modelo relacional. Dados complexos e semi-estruturados não são perfeitamente representados pela modelagem relacional tradicional, e deste modo seu armazenamento em sistemas de bancos de dados relacionais, ainda que possível, pode não ser a melhor opção.

Sistemas de bancos de dados nativos em XML oferecem uma alternativa para esse problema. Para lidar com documentos XML sem o prejuízo em termos de desempenho que mapeamentos para tabelas relacionais revelam, estes sistemas implementam uma granularidade mais “grossa” ao utilizarem o próprio documento como menor unidade de armazenamento lógico.

Procuramos com este trabalho apresentar as alternativas mais utilizadas para o armazenamento e recuperação de informações no formato XML. Aplicativos de middleware têm utilização bastante específica, mas podem ser úteis em determinados contextos. Os

sistemas de bancos de dados relacionais continuam a ser a opção mais robusta para grandes volumes de dados, e após adicionarem formas de suporte ao padrão XML, passaram a ser uma opção interessante também para armazenamento de XML. Isto é válido desde que se observe que, no caso da XML, haverá perda de funcionalidade e desempenho se não se tratar de dados bem estruturados – tabelas relacionais mapeadas para XML, mais especificamente. Já dados em XML semi-estruturados ou representando estruturas complexas (dados aninhados, representações de grafos, dentre outros) encontram uma melhor opção para armazenamento e gerenciamento ao utilizarem sistemas projetados desde o início para lidar com a tecnologia XML: os sistemas de bancos de dados nativos em XML.

Com o objetivo de expor mais sobre esta nova tecnologia, oferecemos a implementação de um banco de dados utilizando-se do XÍndice. Ainda que bastante simples, esta implementação permitiu comprovarmos na prática algumas das características positivas (como o bom desempenho ao lidar com XML) e negativas (tecnologia em fase de maturação, por exemplo) dos sistemas nativos em XML.

Cabe aqui deixar algumas características de sistemas que poderiam ser implementados sob um banco de dados XML nativo:

- Sistemas de pequeno a médio porte, com pequeno volume de dados e transações;
- Sistemas para dispositivos com poucos recursos, como palmtops e handhelds, bem como computadores mais antigos que não suportariam um grande SGBD;
- Interface de integração entre sistemas de back-end e front-end ou outros. Poderíamos ter toda a base de dados relacionada a integração entre sistemas armazenadas nesse tipo de banco de dados, ao invés de utilizar arquivos de configuração e de sincronismo;
- Implementação do próprio sistema de front-end de uma grande aplicação. Poderíamos replicar os dados necessários de um grande SGBD para o XÍndice e

utilizar esses dados para um sistema específico;

- Implementação de sistemas onde os dados a serem armazenados são semi-estruturados, ou seja, não possuem “colunas” fixas, e essas podem ser bastante variadas. Assim, mapear esses dados em um modelo relacional poderia ser bastante trabalhoso, além de desperdiçar espaço (teríamos uma grande quantidade de colunas armazenando valores nulos). Para esses casos, o XML, e mais especificamente o Xindice poderiam ser muito mais indicados, pois a estrutura de um documento XML, apesar de ter sua estrutura, é muito mais flexível e se adaptaria a essa situação facilmente.

Deve-se levar em conta que bancos de dados com grandes volumes de informações e com exigências de robustez e confiabilidade para seu processamento ainda estão limitados aos sistemas relacionais e objeto-relacionais, que implementam as propriedades ACID de processamento de transações.

Por fim, acreditamos que a área de bancos de dados XML (nativos ou não) tem muito para oferecer e tende a estar cada vez mais presente no desenvolvimento de sistemas. Com nossa implementação pudemos perceber que o desenvolvimento de uma aplicação desse tipo é fácil, rápido, e seu desempenho atende perfeitamente às necessidades de aplicações.

REFERÊNCIAS BIBLIOGRÁFICAS

- ALBERTIN, Alberto Luiz. Comércio eletrônico: modelo, aspectos e contribuições de sua aplicação. São Paulo: Atlas, 2001. 3 ed.
- BOURRET, Ronald. XML and Databases [artigo técnico]. Online, disponível em <http://www.rpbouret.com/xml/XMLAndDatabases.htm> . Publicado em janeiro de 2003; último acesso em abril de 2003.
- DATE, C.J. Introdução a sistemas de Bancos de dados. Rio de Janeiro: Campus, 2000.
- DAUM, B. MERTEN, U. Arquiteturas de sistemas com XML. Rio de Janeiro: Campus, 2002.
- ELMASRI, R.; NAVATHE, S. B. Fundamentals of database systems. Addison-Wesley, c2000, 3 ed.
- ENNSER, L; DELPORTE, C.; OBA, M.; SUNIL, K. M. Integrating XML with DB2 XML Extender and DB2 Text Extender.
Online, disponível em www.redbooks.ibm.com/redbooks/SG246130.html .
Direitos de cópia de 2000; último acesso em maio de 2003.
- HAAS, L. M.; LIN, E. T.; ROTH, M. A. Data integration through database federation [artigo técnico]. Disponível em <http://www.research.ibm.com/journal/sj/414/haas.pdf> . Publicado em 2002; último acesso em maio de 2003.
- KROENKE, David M. Database processing: fundamentals, design & implementation. Prentice-Hall, 7 ed., 2000.
- MCGOVERAN, David. The Age of XML Database [artigo técnico]. Online, disponível em <http://www.eaijournal.com/PDF/XMLMcGoveranbb.pdf> . Publicado em 2001, último acesso em maio de 2003.
- MUENCH, Steve. Building Oracle XML Applications. O'Reilly & Associates, 2000.

NAMBIAR, U.; LACROIX, Z.; BRESSAN, S.; LEE, M. L.; LI, Y. Current approaches to XML Management in IEEE Internet Computing Magazine (jul/ago de 2002).

OBASANJO, Dare. An exploration of XML database management systems [artigo técnico]. Disponível em <http://www.25hoursaday.com/StoringAndQueryingXML.html> Publicado em 2001; último acesso em maio de 2003.

OS REIS do E-commerce in Info Exame, São Paulo, n. 206, abr. 2003.

SILBERSCHATZ, A.; KORTH, H.; SUDARSHAN, S. Sistemas de bancos de dados. São Paulo: Makron Books, 1999. 3 ed.

STAKEN, Kimbro. XIndice User's Guide 0.7 . Manual online do produto XIndice, disponível em <http://xml.apache.org/XIndice/UsersGuide.html> . Publicado em março de 2002; último acesso em abril de 2003.

VIANU, Victor. A Web Odyssey: from Codd to XML [artigo técnico]. Disponível em <http://citeseer.nj.nec.com/448486.html> . Publicado em 2001; último acesso em abril de 2003.

WALSH, Norman. Appendix: Extended Backus-Naur Form (EBNF) [artigo técnico]. Disponível em <http://www.xml.com/pub/a/98/10/guide5.html> . Publicado em outubro de 2001; último acesso em junho de 2003.

WILLIAMS, Kevin. Native XML databases: a bad idea for data? [artigo técnico]. Disponível em <http://www-106.ibm.com/developerworks/xml/library/x-xdnat.html> . Publicado em outubro de 2001; último acesso em abril de 2003.

XML: DB INITIATIVE. Frequently asked questions about XML:DB. Disponível em www.xmldb.org/faqs.html . Último acesso em março de 2003.

BIBLIOGRAFIA COMPLEMENTAR

- AHMED, K. Z.; UMRYSH, C. E. Desenvolvendo aplicações comerciais em Java com J2EE e UML. Ciência Moderna, 2002.
- DEEPAK, ALUS, Core J2EE Patterns – as melhores práticas e estratégias de design. Campos, 2002.
- GEARY, David M. Java Server Pages avançado. Ciência Moderna, 2002.
- GRAVES, Mark. Projeto de banco de dados com XML. Makron Books, 2003.
- HOSTMANN, Cay S.; CORNELL, Gary. Core Java 2: fundamentos. Makron Books, 2000.
- HOSTMANN, Cay S.; CORNELL, Gary. Core Java 2: recursos avançados. Makron Books, 2000.
- MARCHAL, B. XML conceitos e aplicações. Berkeley, 2000.
- PITTS-MOULTIS, N.; KIRK, C. XML Black Book. Makron Books, 2000.
- RAY, Erik T. Aprendendo XML. Rio de Janeiro: Campus, 2001.
- TIDWELL, Doug. Introduction to XML [tutorial]. Online, disponível em www-106.ibm.com/developerworks/xml. Último acesso em fevereiro de 2003. .

APÊNDICE A – CÓDIGOS FONTE DA IMPLEMENTAÇÃO

Esse apêndice compreende alguns códigos fonte para ilustrar como foi feita a implementação do sistema utilizando a linguagem Java com o banco de dados Xindice. O código fonte completo da aplicação está disponível na Universidade Mackenzie, na Faculdade de Computação e Informática.

Arquivo empresaBean.java

Classe que representa a entidade Empresa do sistema.

```
package Beans;

/**
 * @author Flávio Jimenez, Hilton Higasi, Lincoln Masao Muraoka Junior
 *
 */

import java.util.*;

public class empresaBean {

    private String docID;
    private String empID;
    private String razao;
    private String endereco;
    private String telefone;

    public void setDocID(String DOCID){
        this.docID = DOCID;
    }

    public void setEmpID(String EMPID){
        this.empID = EMPID;
    }

    public void setRazao(String RAZAO){
        this.razao = RAZAO;
    }

    public void setEndereco(String END){
        this.endereco = END;
    }

    public void setTelefone(String TEL){
        this.telefone = TEL;
    }

    public String getDocID(){
        return this.docID;
    }

    public String getEmpID(){
        return this.empID;
    }

    public String getRazao(){
        return this.razao;
    }

    public String getEndereco(){
        return this.endereco;
    }

    public String getTelefone(){
        return this.telefone;
    }
}
```

```

    }

    public empresaBean(){
        this.empID = "";
        this.razao = "";
        this.endereco = "";
        this.telefone = "";
    }

    public empresaBean(String EMPID, String RAZAO,String END,String TEL){
        this.setEmpID(EMPID);
        this.setRazao(RAZAO);
        this.setEndereco(END);
        this.setTelefone(TEL);
    }
}

```

Arquivo Action.java

Interface que define os métodos de todas as ações do sistema

```

package Action;

/**
 * @author Flávio Jimenez, Hilton Higasi, Lincoln Masao Muraoka Junior
 *
 */

import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public interface Action {

    public ActionRouter perform(HttpServlet servlet,
                                HttpServletRequest req,
                                HttpServletResponse res)
        throws java.io.IOException,
        javax.servlet.ServletException;

}

```

Arquivo addContato.Java

Classe de ação responsável por cadastrar um novo contato a uma empresa no banco de dados

```
package Action;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import Beans.*;
import Core.*;

/**
 * @author Flávio Jimenez, Hilton Higasi, Lincoln Muraoka
 *
 */

public class addContato implements Action {

    private DBConnection connection = new DBConnection();

    public ActionRouter perform(
        HttpServlet servlet,
        HttpServletRequest req,
        HttpServletResponse res)
        throws IOException, ServletException {

        Collection col = null;

        String xmlUpdate = "";
        xmlUpdate += "<xupdate:modifications version=\"1.0\" _";
        xmlUpdate += "xmlns:xupdate=\"http://www.xmldb.org/xupdate\">";
        xmlUpdate += "<xupdate:append select=\"/Empresa[@empID='\" + _";
        xmlUpdate += req.getParameter("docID") + "']/contatos\">";
        xmlUpdate += "<xupdate:element name=\"Contato\">";
        xmlUpdate += "<xupdate:attribute name=\"contID\">\" + _";
        xmlUpdate += req.getParameter("contID") + "</xupdate:attribute>";
        xmlUpdate += "<xupdate:element name=\"nome\">\" + _";
        xmlUpdate += req.getParameter("nome") + "</xupdate:element>";
        xmlUpdate += "<xupdate:element name=\"cargo\">\" + _";
        xmlUpdate += req.getParameter("cargo") + "</xupdate:element>";
        xmlUpdate += "<xupdate:element name=\"telefone\">\" + _";
        xmlUpdate += req.getParameter("telefone") + "</xupdate:element>";
        xmlUpdate += "<xupdate:element name=\"ligacoes\"></xupdate:element>";
        xmlUpdate += "</xupdate:element>";
        xmlUpdate += "</xupdate:append>";
        xmlUpdate += "</xupdate:modifications>";
```

```

try{

    col = connection.getCollectionInstance();
    XUpdateQueryService service = (XUpdateQueryService) _
        col.getService("XUpdateQueryService","1.0");
    service.update(xmlUpdate);

    String msg;
    msg = "Contato adicionado com sucesso!";
    req.getSession().setAttribute("mensagem",new sucessBean(msg));
    return new ActionRouter("/sucesso.jsp");

}catch(XMLDBException e){

    String err;
    err = "Erro do XIndice";
    req.getSession().setAttribute("erro",new errorBean(err));
    return new ActionRouter("/erro.jsp");

}catch(Exception e){

    req.getSession().setAttribute("erro",new errorBean("Exceção"));
    return new ActionRouter("/erro.jsp");

}
}
}

```

Arquivo searchContato.java

Classe de ação responsável por efetuar busca de empresas no banco de dados

```

package Action;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.*;
import org.xmldb.api.base.*;
import org.xmldb.api.modules.*;
import org.w3c.dom.*;
import org.apache.xpath.*;
import org.apache.xpath.objects.*;
import Core.*;
import Beans.*;

/**
 * @author Flávio Jimenez, Hilton Higasi, Lincoln Muraoka
 *
 */

public class searchContato implements Action {

```

```

private DBConnection connection = new DBConnection();

public ActionRouter perform(
    HttpServletRequest servlet,
    HttpServletRequest req,
    HttpServletResponse res)
    throws IOException, ServletException {

    org.xmldb.api.base.Collection col = null;

    String Query = "";

    if(req.getParameter("tipoConsulta") == null){

        Query = "/Empresa/Contatos/contato";

    }else{
        String tipoConsulta = req.getParameter("tipoConsulta");
        String criterio = req.getParameter("criterio");
        Query = "/empresa/Contatos/contato[" + tipoConsulta + _
            "'=" + criterio + "']";
    }

    ArrayList contatos = new ArrayList();

    try{

        col = connection.getCollectionInstance();
        XPathQueryService service = (XPathQueryService) _
            col.getService("XPathQueryService", "1.0");

        ResourceSet resultSet = service.query(Query);
        ResourceIterator results = resultSet.getIterator();

        while(results.hasMoreResources()){

            contatoBean contato = new contatoBean();

            MLResource resource = (XMLResource) _
                results.nextResource();
            Node originalnode = resource.getContentAsDOM();

            XObject xo = new XObject();

            xo = XPathAPI.eval(originalnode, "/contato/@codCont");
            contato.setCodCont(xo.toString());

            xo = XPathAPI.eval(originalnode, "/contato/nome");
            contato.setNome(xo.toString());

            xo = XPathAPI.eval(originalnode, "/contato/cargo");
            contato.setCargo(xo.toString());

            xo = XPathAPI.eval(originalnode, "/contato/telefone");
            contato.setTelefone(xo.toString());

            contatos.add(contato);

        }

        req.getSession().setAttribute("contatos", contatos);
    }
}

```



```

return new ActionRouter("/listContato.jsp");

}catch(XMLDBException e){

    String errX;
    errX = "Erro do XIndice";
    req.getSession().setAttribute("erro",new errorBean(errX));
    return new ActionRouter("/erro.jsp");

}catch(Exception e){

    req.getSession().setAttribute("erro",new _
        errorBean(e.toString()));
    return new ActionRouter("/erro.jsp");

}

}

}

```

Arquivo listEmpresa.jsp

Página web que serve de front-end para a consulta de empresas.

```

<%@page import="java.util.*"%>
<%@page import="Beans.*"%>
<HTML>
<HEAD>
<link rel="stylesheet" href="estilo.css" type="text/css">
<META NAME="GENERATOR" Content="Microsoft Visual Studio 6.0">
<TITLE></TITLE>
</HEAD>
<BODY scroll=no topmargin=0 leftmargin=0 rightmargin=0>
<br><br>
<center>
<table border=0 width=650 cellpadding=2 cellspacing=1 bgcolor=17717E>
<jsp:useBean id="empresas" scope="session" class="ArrayList" />
    <tr>
        <td colspan=2 align=center>
            <font face=verdana size=1 color=white>
                <b>
                    Consulta de Empresas
                </b>
            </font>
        </td>
    </tr>
<%
    if(empresas != null){
%>

<form action="Action.searchEmpresa.do">
<tr>
    <td colspan=2>
        <table bgcolor=white border=0 cellpadding=0 cellspacing=0
width=100%>
            <tr>

```

```

        <td align=center>
            <font face=verdana size=1 color=white>
                <b>
                    Critério:
                    <select name="tipoConsulta">

                        <option value="empID">Codigo da Empresa</option>

                        <option value="razao">Razão social</option>

                        <option value="endereco">Endereço</option>

                        <option value="telefone">telefone</option>
                    </select>
                </b>
            </font>
        </td>
        <td align=center>
            <input type=text name=criterio><input type=submit
value="ok">
        </td>
    </tr>
</table>
</td>
</tr>
</form>
<tr bgcolor=white height=20>
    <td colspan=2>
</tr>
<tr bgcolor=white>
    <td width=115>
        <font face=verdana size=1 color=17717E>
            <b>
                Código da Empresa
            </b>
        </font>
    </td>
    <td width=435>
        <font face=verdana size=1 color=17717E>
            <b>
                Razão Social
            </b>
        </font>
    </td>
</tr>
<%
    for(int i = 0; i < empresas.size(); i++){
        empresaBean empresa = (empresaBean)empresas.get(i);
    %>
<tr bgcolor=white>
    <td>
        <font face=verdana size=1 color=black>
            <%=empresa.getEmpID()%>
            <a style="text-decoration:none"
href="javascript:PopUp( 'Action.getEmpresa.do?docID=
<%=empresa.getDocID()%>&page=detEmpresa.jsp',
'width=520,height=250,scrollbars=yes' )" ><b>+</b>
            </a>
        </font>
    </td>
    <td>

```

```

                <font face=verdana size=1 color=black>
                    <%=empresa.getRazao()%>
                </font>
            </td>
        </tr>
        <%
            }
        %>

        <%
            }else{
        %>
        <tr bgcolor=white>
            <td colspan=2 align=center>
                <font face=verdana size=1 color=red>
                    <b>
                        Atenção. Não existem empresas cadastradas.
                    </b>
                </font>
            </td>
        </tr>

        <%
            }
        %>
    </table>
</center>
</BODY>
</HTML>
<script>
function PopUp(txt,opc)
{
    pop=window.open(txt,'pop2',opc);
    pop.focus();
}
</script>

```

FLÁVIO JIMENEZ VÉRDI DE FIGUEIREDO
HILTON HIGASI
LINCOLN MASAO MURAOKA JUNIOR

XML E BANCOS DE DADOS

Trabalho apresentado à disciplina de
Trabalho de Graduação Interdisciplinar II,
como parte das exigências para a obtenção
do título de Bacharel em Sistemas de Informação
pela Faculdade de Computação e Informática
da Universidade Presbiteriana Mackenzie

ORIENTADOR: ROGÉRIO DE OLIVEIRA

São Paulo
2003

RESUMO

A XML (eXtensible Markup Language) foi desenvolvida como um subconjunto da SGML (Standard Generalized Markup Language). Ela foi concebida como uma opção flexível e portátil para representação e intercâmbio de dados pela internet.

A adoção da XML como um padrão por diversas áreas de negócios vem crescendo cada vez mais. Este crescimento traz consigo um desafio: como lidar com o grande volume de dados que é gerado e transportado usando XML.

Novas tecnologias de bancos de dados voltadas para manipulação de dados XML disputam espaço no mercado com os grandes sistemas de bancos de dados relacionais, apresentando características específicas para o processamento de XML. Os bancos relacionais, por sua vez, oferecem extensões para XML em seus sistemas tentando se adaptar satisfatoriamente a essa nova tecnologia de representação de dados.

Este trabalho analisa possíveis soluções para o problema do armazenamento e recuperação de dados no formato XML. Também apresenta uma implementação de banco de dados usando um sistema de banco de dados nativo em XML, mais especificamente o sistema Xindice, para ilustrar na prática o que se pode esperar atualmente do uso desta abordagem de armazenamento para documentos XML.

ABSTRACT

The XML (eXtensible Markup Language) was developed as a subset of the SGML (Standard Generalized Language). It was conceived as a flexible and portable choice for data representation and exchange through the internet.

The adoption of the XML as a standard language by many bussines areas is growing day after day. But this growth also brings a challenge with it: how to deal with the large amount of data created and transportated by the XML?

New database technologies in the data manipulation field are beginning to compete for a share of the market with the big relational database systems, presenting specific characteristics for the XML processing. The relational database systems, on the other hand, offer XML extensions in their systems, trying to adapt themselves satisfactorily to this new data representation technology.

This work analyses possible solutions for the XML data storing and retrieving issue, presenting its main points. It also shows a database implementation using a native XML database system, more specifically the Xindice, to illustrate in practice what one can expect nowadays from the use of this approach to the XML document storage.