UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ESTUDO COMPARATIVO SOBRE BANCOS DE DADOS XML NATIVO E DESENVOLVIMENTO DE UMA APLICAÇÃO

TRABALHO DE GRADUAÇÃO

Mauro Guilherme de Assumpção Marinho

Santa Maria, RS, Brasil

2012

ESTUDO COMPARATIVO SOBRE BANCOS DE DADOS XML NATIVO E DESENVOLVIMENTO DE UMA APLICAÇÃO

por

Mauro Guilherme de Assumpção Marinho

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de
Bacharel em Ciência da Computação

Orientador: Profª Drª Deise de Brum Saccol

Trabalho de Graduação Nº 344
Santa Maria, RS, Brasil

2012

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

ESTUDO COMPARATIVO SOBRE BANCOS DE DADOS XML NATIVO E DESENVOLVIMENTO DE UMA APLICAÇÃO

elaborado por
Mauro Guilherme de Assumpção Marinho

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Prof[a] Dr[a] Deise de Brum Saccol
(Presidente/Orientador)

Prof Dr Eduardo Kessler Piveta

Prof Dr Giovani Rubert Librelotto

Santa Maria, 04 de julho de 2012.

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

ESTUDO COMPARATIVO SOBRE BANCOS DE DADOS XML NATIVO E DESENVOLVIMENTO DE UMA APLICAÇÃO

Autor: Mauro Guilherme de Assumpção Marinho
Orientador: Profª Drª Deise de Brum Saccol
Local e data da defesa: Santa Maria, 04 de julho de 2012.

A linguagem XML está cada vez mais conhecida nos últimos anos. Além de ela ser muito utilizada para armazenamento e troca de informações, é também uma linguagem de fácil entendimento, por duas razões: a primeira é que um documento pode ser codificado em XML ainda assim será possível ler o documento e entendê-lo; a segunda é que um documento XML pode representar informações de forma hierárquica, tornando possível a inserção de informações ao seu conteúdo.

Entretanto, apesar de XML ser uma linguagem bem estruturada, devido ao tamanho e ao número de *tags* que um documento poder ter, torna-se complicado buscar informações em uma coleção de documentos XML sem o auxílio de uma ferramenta especializada, como por exemplo, um banco de dados. Desse modo, o objetivo deste trabalho é realizar uma pesquisa, selecionando alguns bancos de dados XML nativos (NXD) e compará-los entre si, mostrando suas funcionalidades comuns e específicas. Após essa comparação, será escolhido um NXD, através de suas características de funcionamento, para o desenvolvimento de uma aplicação, que utilizará o próprio NXD para consultas e modificações dos dados de sua base de dados.

Palavras-chave: XML, XML Nativo, Banco de Dados.

ABSTRACT

Undergraduate Final Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

COMPARATIVE STUDY ON NATIVE XML DATABASE AND DEVELOPMENT OF AN APPLICATION

Author: Mauro Guilherme de Assumpção Marinho
Advisor: Profª Drª Deise de Brum Saccol

The XML language is becoming increasingly known in recent years. Besides being widely used for storing and exchanging information, it is also a language easy to understand for two reasons: first, a document may be encoded in XML and you can still read and understand it; second, a XML document can represent information in a hierarchical manner, making it possible to insert information to your content.

However, despite the fact that the XML is a well structured language, it becomes difficult to find information in a collection of documents without the help of a specialized tool, such as a database. Thus, the objective of this work is to perform a search, select some native XML databases (NXD) and compare them, showing common and specific features. After this comparison, to choose a NXD and develop an application that uses the NXD for querying and modifying data into the database.

Keywords: XML, Native XML, database.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

DBLP	Digital Bibliography & Library Project
DOM	Document Object Model
DTD	Document Type Definition
GML	Generalized Markup Language
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JEE	Java Enterprise Edition
JDK	Java Development Kit
JVM	Java Virtual Machine
JSP	Java Server Pages
NXD	Native XML Database
SGBD	Sistema de Gerenciamento de Banco de Dados
SGML	Standard Generalized Markup Language
SSL	Secure Sockets Layer
W3C	World Wide Web Consortium
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

As linguagens de marcação tiveram início com a linguagem *Generalized Markup Language* (GML), criada na década de 60 com o intuito de ser utilizada para dar suporte a aplicações de processamento de documentos. Na década de 70, a linguagem *Standard Generalized Markup Language* (SGML) foi criada e entre 1978 e 1980 foi aperfeiçoada e desenvolvida pela *International Standards Organization* (ISO) para atender requisitos de independência do sistema e troca de informações internacionais.

A *Hypertext Markup Language* (HTML), derivada da SGML, é baseada em *tags* e foi desenvolvida nos anos 80. A HTML é simples e se tornou muito popular por sua utilização na Internet. Porém, não é muito favorável à transferência de dados, pois ela mistura o conteúdo com a sua representação.

A SGML foi projetada para permitir a separação do conteúdo e representação, porém é muito complexa. Assim, em 1996, iniciou-se o desenvolvimento da *Extensible Markup Language* (XML), também baseada em *tags*. Seria uma solução para o problema de complexidade e de transferência de dados.

A XML permite separar o conteúdo da formatação de um documento de forma simples. Também é possível efetuar troca de informações entre aplicações ou entre aplicações e usuários. (GRAVES, 2002).

Diferente da HTML, a XML não possui controle de quais *tags* poderão ser criadas em um documento, o usuário é quem define as *tags* utilizadas. Para defini-las, foram criadas as linguagens *Document Type Definition* (DTD) e XML Schema. Para que a manipulação dos arquivos XML fosse feita de forma mais simples, foi criado o *Document Object Model* (DOM), que auxilia no acesso das informações de um arquivo.

A utilização de XML tem aumentado em larga escala para a transmissão e armazenamento de dados. Portanto, o objetivo deste trabalho é estudar alguns bancos de dados XML nativos disponíveis para utilização e desenvolver uma aplicação para um deles, a

fim de difundir outras técnicas de armazenamento de dados além do armazenamento através de bancos de dados relacionais.

1.1 JUSTIFICATIVA

Documentos XML são amplamente utilizados para troca e/ou armazenamento de informações. Muitas vezes esses documentos podem ser muito grandes e com muitas *tags*, o que torna difícil o acesso rápido às informações armazenadas. Para que esse acesso às informações seja ágil e seguro, podem-se usar bancos de dados XML nativo, que podem manter os arquivos armazenados, no próprio formato XML, de forma ordenada e acessá-los através de índices, dependendo de como os dados foram armazenados. Sendo assim, uma pesquisa será realizada para selecionar alguns bancos de dados XML nativo e, após uma comparação entre eles, será escolhido um para o desenvolvimento de uma aplicação.

1.2 OBJETIVOS

O objetivo geral deste trabalho é fazer uma pesquisa e comparação de alguns *Bancos de Dados XML Nativos* (NXD – *Native XML Database*) e desenvolver uma aplicação que use um NXD selecionado. Como objetivos específicos, podem ser citados:

- Pesquisa e seleção de NXDs;
- Estudo individual dos NXDs e das linguagens de consulta;
- Comparação dos NXDs e eleição de um deles para desenvolver a aplicação;
- Definição da aplicação a ser desenvolvida;
- Desenvolvimento da aplicação;
- Testes da aplicação desenvolvida;
- Elaboração da parte escrita.

1.3 ORGANIZAÇÃO DO TEXTO

A organização deste texto apresenta-se da seguinte forma.

O Capítulo 2 apresenta uma revisão de literatura sobre alguns conceitos envolvendo a linguagem XML, uma breve apresentação do *Digital Bibliography & Library Project*

(DBLP), o padrão *Document Object Model* (DOM), bem como as linguagens Definição de Tipo de Documento (DTD) e XML Schema, que são utilizadas para validação de documentos XML. Após, são apresentados os bancos de dados XML nativos estudados neste trabalho, bem como as linguagens de consulta de cada banco.

O Capítulo 3 contém a proposta da ferramenta desenvolvida, explicando quais as funcionalidades da aplicação, que irá interagir com um NXD. Também é apresentado o NXD selecionado para realizar o desenvolvimento da aplicação e o motivo pelo qual ele foi escolhido. Em adição, este capítulo mostra um diagrama de casos de uso da ferramenta.

O Capítulo 4 explica como foi desenvolvida a ferramenta proposta, com uma breve explicação sobre a implementação. Também são comentadas quais as tecnologias utilizadas para o desenvolvimento (algumas imagens são exibidas para ajudar na explicação do desenvolvimento e utilização da ferramenta).

Por fim, o Capítulo 5 encerra o trabalho com as considerações finais e sugestões de melhorias para trabalhos futuros.

2 REVISÃO DE LITERATURA

Este capítulo apresenta algumas tecnologias, tais como: linguagem XML; linguagens para validação de documentos XML, como DTD e XML Schema; DBLP, que é um repositório de documentos relacionados a publicações na área da Computação; o padrão DOM; e por fim, os NXDs selecionados para este trabalho.

2.1 XML

A XML é uma linguagem de marcação de documentos semelhante à linguagem HTML. A XML é um subtipo da SGML com a capacidade de descrever diversos tipos de dados (W3C, 2012).

A XML foi inicialmente projetada para tratar problemas como publicações eletrônicas de larga escala através do isolamento de conteúdo por formatação, separando o conteúdo de um documento e formatando-o em formato XML.

A estrutura da linguagem é baseada em elementos formados por um par de *tags* com o mesmo nome. Eles indicam o começo e o fim de cada elemento. Para um texto estar no contexto de um elemento, ele deve estar entre a *tag* onde o elemento começa e a *tag* onde ele termina. A figura 2.1 exibe um exemplo de codificação XML.

Uma das vantagens da XML é que se podem representar documentos muito complexos de forma razoável, tornando sua visualização ou pesquisa mais acessíveis.

```
<?xml version="1.0" encoding="UTF-8"?>
<dblp>
    <article key="journals/pvldb/Jagadish11a" mdate="2011-10-18">
        <author>H. V. Jagadish</author>
        <title>Letter from the Founding Editor-in-Chief.</title>
        <pages>vii</pages>
        <year>2011</year>
        <volume>5</volume>
        <journal>PVLDB</journal>
        <number>1</number>
        <ee>http://vldb.org/pvldb/vol5/frontmatterVol5No1.pdf</ee>
        <url>db/journals/pvldb/pvldb5.html#Jagadish11a</url>
    </article>
</dblp>
```

Figura 2.1 - Documento "Jagadish11a.xml" (DBLP, 2012).

Por outro lado, a sintaxe da XML é redundante e torna-se grande em relação a representações de dados semelhantes, ou seja, é desnecessário usar uma estrutura para transmitir poucos dados ou dados simples. A figura 2.2 é um exemplo de redundância desnecessária.

```
<dados>
    <itens id="1">
        <valor>10</valor>
    </itens>
    <itens id="2">
        <valor>20</valor>
    </itens>
    <itens id="3">
        <valor>30</valor>
    </itens>
</dados>
```

Figura 2.2 - Exemplo de uso desnecessário da linguagem XML.

Agora suponha que milhares de itens serão transmitidos com a mesma formatação da figura 2.2. A quantidade de informação real seria muito menor que a quantidade de informações extras contidas no arquivo, nesse caso as *tags*.

Outra questão importante é que XML não possui mecanismos para criação de regras de validação, isto é, não é possível definir quais blocos são válidos no documento. Para isso, utilizam-se as linguagens DTD e XML Schema.

A DTD define quais *tags* podem ser usadas em um documento XML e quais são os valores válidos. Assim, um programador não precisa se preocupar com a validação do documento e a integridade dos dados – a DTD assumirá esta responsabilidade.

Apesar de ser uma linguagem simples, a DTD possui algumas limitações como: imposição na ordem de ocorrência de elementos, tipos de dados limitados, um documento XML não pode estar associado a mais de uma DTD, dentre outras limitações.

Uma forma de contornar a maioria dos problemas da DTD é a utilizar XML Schema. XML Schema tem como objetivo validar documentos XML e foi criada para dar maior suporte à validação de documentos XML. (Wikipedia, 2012)

A XML Schema disponibiliza muitas funcionalidades. Através dela é possível utilizar tipos de dados como *string*, *date*, entre outros. Essa linguagem também possui maior controle sobre os elementos que devem aparecer em um documento XML do que a DTD. Além disso, XML Schema possibilita a reutilização de código, oferece suporte à construção de tipos próprios derivados dos tipos básicos, realiza relacionamentos entre elementos de dados dentro de XML, dentre outras funcionalidades (W3C, 2012).

2.2 DBLP

O DBLP é um repositório bibliográfico na área de Computação localizado na *Universität Trier*, na Alemanha. Originalmente, o DBLP era chamado de *DataBase systems and Logic Programming*.

Atualmente o site do DBLP possui um número muito grande de artigos sobre computação armazenados em sua base de dados. Ele também armazena publicações de várias conferências, revistas, séries, entre outras.

Uma das formas de armazenamento desses documentos é o armazenamento em formato XML. Os documentos podem ser acessados e baixados através do site do DBLP (http://dblp.uni-trier.de/db/index.html).

O DBLP utiliza uma DTD padrão para validar o armazenamento dos documentos no formato XML. O documento "*dblp.dtd*" pode ser encontrado no site do DBLP, informado no parágrafo anterior. A figura 2.3 mostra alguns trechos do documento "*dblp.dtd*".

O documento DTD do DBLP é muito extenso para ser apresentado por completo, portanto, para melhor visualizá-lo é necessário fazer seu download no site do DBLP.

```
<!ELEMENT dblp (article|inproceedings|proceedings|book|incollection|
                phdthesis|mastersthesis|www)*>
<!ENTITY % field "author|editor|title|booktitle|pages|year|address|journal|volume|number|
          month|url|ee|cdrom|cite|publisher|note|crossref|isbn|series|school|chapter">

<!ELEMENT article       (%field;)*>
<!ATTLIST article
                        key CDATA #REQUIRED
                        reviewid CDATA #IMPLIED
                        rating CDATA #IMPLIED
>
...
<!ELEMENT proceedings   (%field;)*>
<!ATTLIST proceedings   key CDATA #REQUIRED>
...
<!ELEMENT author    (#PCDATA)>
<!ELEMENT editor    (#PCDATA)>
<!ELEMENT address   (#PCDATA)>

<!ENTITY % titlecontents "#PCDATA|sub|sup|i|tt|ref">
<!ELEMENT title     (%titlecontents;)*>
<!ELEMENT booktitle (#PCDATA)>
...
<!ELEMENT publisher (#PCDATA)>
<!ATTLIST publisher
...
```

Figura 2.3 - Documento "*dblp.dtd*". (DBLP, 2012)

Mesmo assim ainda é válido ressaltar que o documento define um elemento nó raiz chamado "<*dblp*>". Dentro do elemento raiz do documento XML é possível ser inserido outros nós tais como <*article*> e <*proceedings*>. Esses dois nós elemento ainda podem possuir os nós filhos informados na entidade *"%field*".

A seguir, a figura 2.4 mostra um exemplo de documento XML de um artigo, armazenado no site do DBLP.

```
<?xml version="1.0" encoding="UTF-8"?>
<dblp>
   <article key="journals/pvldb/CormodeTY11" mdate="2011-10-18">
      <author>Graham Cormode</author>
      <author>Justin Thaler</author>
      <author>Ke Yi</author>
      <title>Verifying Computations with Streaming Interactive Proofs.</title>
      <pages>25-36</pages>
      <year>2011</year>
      <volume>5</volume>
      <journal>PVLDB</journal>
      <number>1</number>
      <ee>http://www.vldb.org/pvldb/vol5/p025_grahamcormode_vldb2012.pdf</ee>
      <url>db/journals/pvldb/pvldb5.html#CormodeTY11</url>
   </article>
</dblp>
```

Figura 2.4 - Documento CormodeTY11.xml.

Com a apresentação deste exemplo é possível fazer a comparação das *tags* presentes no documento "*CormodeTY11.xml*", exibido na figura 2.4, com a DTD utilizada pelo DBLP para validar os documentos inseridos em sua base dados, exibida na figura 2.3.

Alguns documentos do DBLP serão utilizados como exemplo neste trabalho e também nos exemplos de armazenamento do NXD escolhido para o desenvolvimento de uma aplicação, que será proposta no Capítulo 3.

2.3 DOM

O DOM é um padrão da *World Wide Web Consortium* (W3C) que define um modelo de acesso a documentos como XML, XHTML e HTML. (W3C, 2012)

Através do DOM é possível alterar e editar de forma dinâmica a estrutura, o conteúdo e o estilo de um documento eletrônico, tornando possível que este documento seja processado e em seguida os resultados desse processamento sejam incorporados de volta ao próprio documento. A API DOM oferece uma maneira padrão de acesso aos elementos de um documento, além da opção de manipular cada elemento separadamente.

DOM é utilizado para definir os objetos e as propriedades de todos os elementos XML e os métodos para acessá-los, ou seja, DOM é um padrão para obter, modificar, adicionar ou remover elementos XML.

Para manipular um documento XML, o DOM constrói, através de um analisador sintático, um modelo de árvore do arquivo. Sendo assim, para o DOM, cada elemento contido em um documento XML é um nó dessa árvore. A figura 2.5 mostra um exemplo de documento XML que será analisado a seguir.

```
<?xml version="1.0" encoding="UTF-8"?>
<dblp>
    <article key="journals/pvldb/Jagadish11a" mdate="2011-10-18">
        <author>H. V. Jagadish</author>
        <title>Letter from the Founding Editor-in-Chief.</title>
        <pages>vii</pages>
        <year>2011</year>
        <volume>5</volume>
        <journal>PVLDB</journal>
        <number>1</number>
        <ee>http://vldb.org/pvldb/vol5/frontmatterVol5No1.pdf</ee>
        <url>db/journals/pvldb/pvldb5.html#Jagadish11a</url>
    </article>
</dblp>
```

Figura 2.5 - Documento *Jagadish11a.xml*, retirado do site do DBLP.

O nó raiz do documento XML expresso na imagem é nomeado *<dblp>*. Todos os outros nós do documento estão contidos dentro do nó raiz. O nó raiz contém um nó *<article>*, que por sua vez contém outros nós nomeados *<atuhor>*, *<title>*, *<pages>*, *<year>*, *<volume>*, *<journal>*, *<number>*, *<ee>* e *<url>* e cada um desses nós contém um nó texto, como por exemplo, "H. V. Jagadish", "Letter from the Founding Editor-in-Chief.", etc.

Um erro comum no processamento DOM é esperar que um elemento nó contenha texto. Entretanto, o texto de um elemento nó está armazenado dentro de um nó texto. Por exemplo, *<year>2011</year>*, o nó elemento *<year>* contém um nó texto com o valor "*2011*", portanto o valor pertence ao nó texto e não ao elemento nó *<year>*.

Como falado antes, o DOM vê o documento XML como uma árvore de nós. Todos esses nós podem ser acessados pela árvore. O conteúdo dos nós pode ser modificado ou removido e também é possível adicionar novos elementos à árvore.

A figura 2.6 mostra o conjunto de nós formado para o documento *Jagadish11a.xml* mostrado na figura 2.5. A árvore começa no nó raiz e se ramifica até os nós texto no nível mais baixo da árvore.

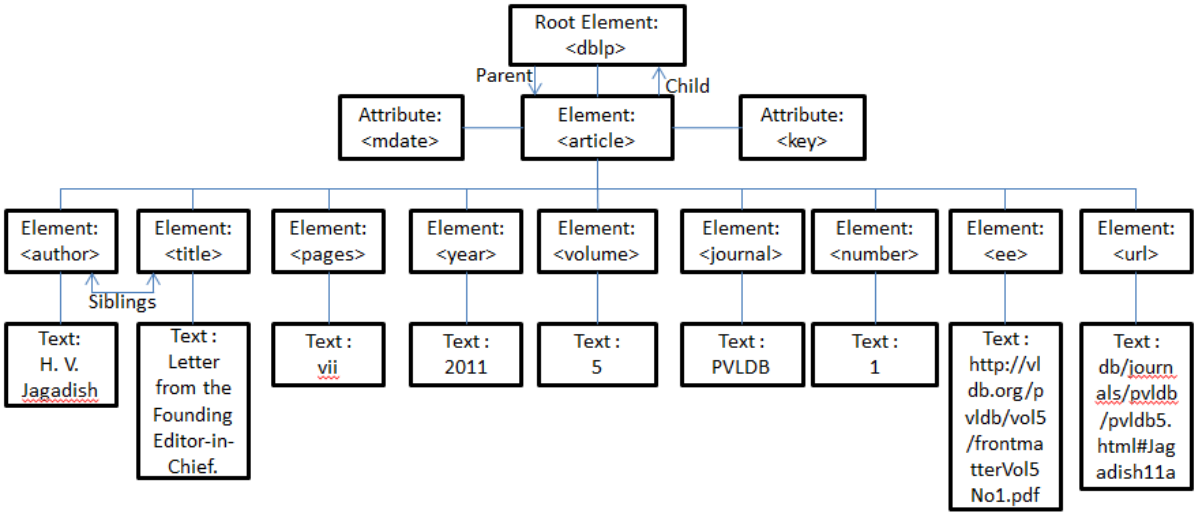

Figura 2.6 - Árvore criada pelo DOM que representa o documento *Jagadish11a.xml*.

Os nós da árvore possuem um relacionamento hierárquico entre eles, e são separados em *parents* (pais), *children* (filhos), *siblings* (irmãos). Estes termos representam a hierarquia, ou seja, o nós pais possuem nós filhos e os nós filhos que estão no mesmo nível são chamados de irmãos. Todos os nós podem ter filhos, exceto os nós folha, que são representados pelos nós texto na árvore da figura 2.6.

2.4 BANCOS DE DADOS XML

Nesta seção encontra-se o estudo dos NXDs pesquisados para a realização deste trabalho.

2.4.1 eXist-db Open Source Native XML Database

O *eXist* é um banco de dados XML nativo de código aberto, desenvolvido em Java. Pode ser instalado nos sistemas operacionais Linux, Windows e Mac OS (eXist, 2012).

2.4.1.1 Arquitetura

O *eXist* pode ser utilizado como um servidor autônomo, embutido em uma aplicação, ou em conexão com um *servlet*. Ele também possui diversas interfaces de acesso, tais como XML-RPC, REST-style Web services API, SOAP, WebDAV e *Atom Publishing Protocol*. A figura 2.7 mostra a arquitetura do *eXist*.

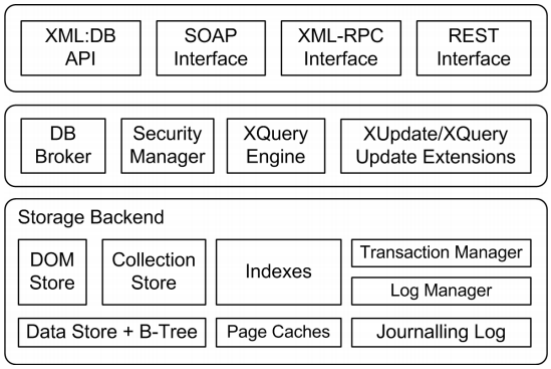

Figura 2.7 - Arquitetura do eXist. (SOUSA, 2007)

No núcleo de sua arquitetura, o *eXist* utiliza quatro arquivos de índice, todos eles baseados em árvore B+. Os elementos, atributos e palavras-chave são organizados por coleção, ao invés de documentos, gerando um considerável ganho de desempenho, tendo em vista que os usuários geralmente consultam coleções inteiras e diferentes simultaneamente. A hierarquia de coleções é gerenciada pelo *Collection Store* que mapeia nomes de coleções para objetos correspondentes. O componente central da arquitetura de armazenamento nativo do *eXist* é o *DOM Store*, que consiste em um arquivo onde todos os nós da árvore de documentos

XML são armazenados. O mapeamento dos nomes de elementos e atributos é feito por um índice que irá identificar cada nó. Há também um índice invertido, que registra ocorrências de palavras e possibilita busca textuais. (SOUSA, 2007)

O *eXist* possui suporte a transações, mas tal suporte é limitado a funcionalidades de recuperações de *crash*, sendo uma desvantagem para este banco de dados, visto que as transações devem ser gerenciadas pela própria aplicação.

O *eXist* possui suporte para as linguagens de consulta XQuery e XPath. Estas linguagens serão abordadas na seção 2.3 deste capítulo.

2.4.1.2 Armazenamento

O armazenamento de dados no *eXist* é baseado em árvores B+ e arquivos paginados de acordo com o modelo DOM. Os arquivos XML são decompostos, e através de um esquema automático de indexação numérica, são atribuídos números aos seus elementos e atributos, sendo armazenados sem esquema através de coleções hierárquicas. Se o usuário preferir e estiver disposto a usar as extensões do *eXist*, ele pode criar a indexação manualmente. (eXist, 2012)

2.4.1.3 Download e execução

O download do *eXist* pode ser feito através do site *http://exist-db.org*. O link para download está na página principal. No site encontram-se também um tutorial para a instalação e documentação bastante completa para ajudar nos primeiros passos na utilização do *eXist*. A figura 2.8 mostra a interface do programa.

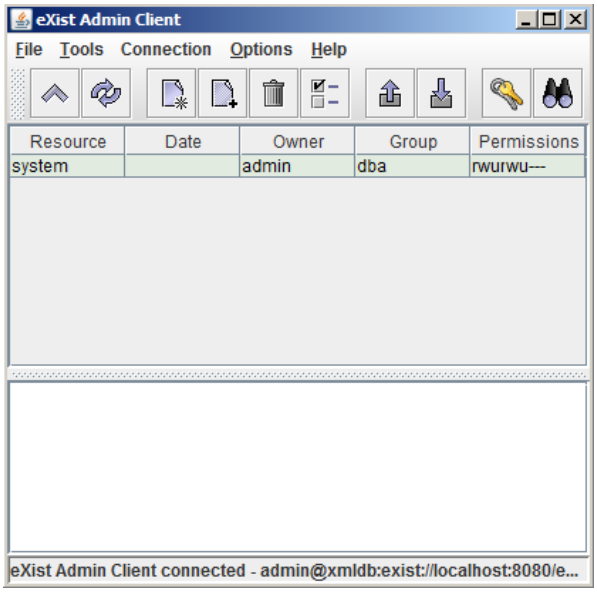

Figura 2.8 - Programa cliente.

Para executar este NXD, pode-se utilizar um programa cliente que faz a interface do banco de dados com o usuário. O programa possibilita criar coleções, armazenar dados, fazer consultas ao banco de dados, etc.

2.4.2 Tamino XML Server

O *Tamino* é um NXD, de licença comercial, desenvolvido e distribuído pela empresa Software AG. No final dos anos 90 ela desenvolveu o *Tamino*, sendo este, um dos primeiros bancos de dados XML nativo. (Blanken, 2003)

O *Tamino* é um servidor de informações que armazena dados XML de múltiplas fontes. Ele armazena, consulta e transforma os dados XML nativamente, utilizando a interface e especificações da web e da linguagem XML. (Tamino, 2002)

2.4.2.1 Arquitetura

O *Tamino* foi desenvolvido em cima do banco de dados ADABAS. Possui estruturas de índices, suporte para tratamento de informações do esquema XML, mecanismo para o processamento de transação e uma camada de interface para a Web (PASQUALI & DUARTE). A figura 2.9 mostra a arquitetura do *Tamino*.

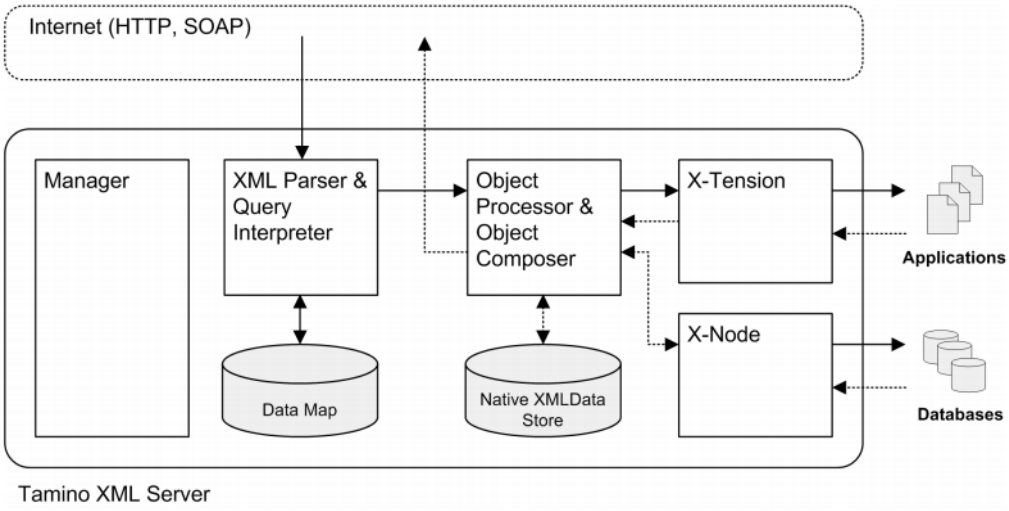

Figura 2.9 - Arquitetura do Tamino. (SOUSA, 2007)

O componente *X-Node* é responsável pelo acesso do banco de dados a dados externos. O componente *X-Tension* é responsável por passar documentos XML para funções definidas pelo usuário ou ler dados das funções informadas para inclusão nos documentos XML. O módulo *Object Processor* é responsável por enviar documentos e comandos ao servidor. Os comandos enviados são então recebidos e avaliados pelo *XML Parser* de acordo com os metadados contidos no repositório *Data Map*.

A comunicação do *Tamino* com servidores externos ou locais é feita através do componente *X-Port*. Esse componente garante uma comunicação eficiente e segura com os servidores, utilizando SSL. Se a aplicação cliente que estiver se comunicando com o *Tamino* também utilizar comunicação SSL, todo o caminho desde a aplicação cliente até o servidor *Tamino* estará seguro.

A estrutura do *Tamino* também oferece suporte direto aos métodos do protocolo de comunicação *Hypertext Transfer Protocol* (HTTP) *GET, POST, DELETE, HEAD* para ler, armazenar, substituir, remover e obter informações de documentos armazenados na base de dados do *Tamino*. (eTutorials, 2012)

O *Tamino* possui sua própria linguagem de consulta, a X-Query, diferente da linguagem de consulta XQuery, que é recomendada pela W3C. A XQuery também é suportada pelo *Tamino*. A X-Query e a XQuery serão abordadas na seção 2.3.

2.4.2.2 Armazenamento

O armazenamento de documentos XML do *Tamino* pode ser interno ou externo. O tipo de armazenamento pode ser definido por um esquema, através do componente *X-Machine*. O armazenamento externo é feito através do componente *X-Node* (Tamino, 2006), e pode ser realizado em SGBDs relacionais. O componente *SQL Engine* cuida do armazenamento interno de dados estruturados, mas também pode ser responsável pelo externo.

O armazenamento de documentos XML não precisa estar associado a um esquema XML, onde estão definidos a forma de armazenamento e o tipo de indexação utilizado em um determinado elemento. A definição de esquema XML só é obrigatória quando o armazenamento de dados é feito externamente, pois o esquema é utilizado para a definição dos parâmetros necessários para o acesso a esses dados. O esquema XML pode ser definido de duas formas: a primeira é manualmente através do *Schema Editor* ou de qualquer editor de texto e a segunda é através da conversão de uma DTD em um esquema *Tamino* correspondente. A linguagem de esquema do *Tamino* é um subconjunto da linguagem *XML Schema* do W3C, pois o *Tamino* usa três tipos de elementos, que são: *collection*, *doctype* e *node*. A figura 2.10 ilustra o armazenamento de dados do *Tamino*.

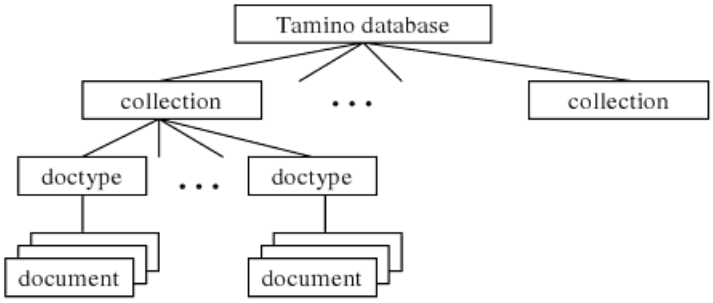

Figura 2.10 - Estrutura de armazenamento de dados do Tamino. (Blanken, 2003)

O banco de dados pode possuir várias coleções, e cada coleção pode possuir vários *doctypes*, ou não ter nenhum.

O *doctype* é um container utilizado para armazenar documentos XML. Eles são agrupados logicamente em conjuntos dentro das coleções. Os tipos de dados que podem ser armazenados no banco de dados devem ser definidos em um XML Schema. Esses tipos de dados serão utilizados posteriormente pelo *doctype* para validar os documentos carregados para armazenamento.

O *node* é um elemento vazio que expressa os itens de informação contidos no *doctype*. O conjunto de *nodes* pode ser organizado na estrutura de árvore, usando atributos para expressar os relacionamentos pai e filho. Assim, alguns nós são intermediários, descrevendo o caminho para o verdadeiro item de informação. [3]

2.4.2.3 Download e execução

O download do Tamino pode ser feito através do seguinte endereço: *http://www.softwareag.com/br/products/wm/tamino/downloads/download.asp*. Para executar o *Tamino* é necessário ter instalado o *Apache* 2.0 ou superior.

A versão 4.4 executa apenas nas seguintes versões do Windows: Windows Server 2000, Windows Server 2003 e Windows XP Professional.

O *Tamino* possui uma interface web interativa, mostrada na figura 2.11, e um sistema de interface desktop chamado *Management Hub*.

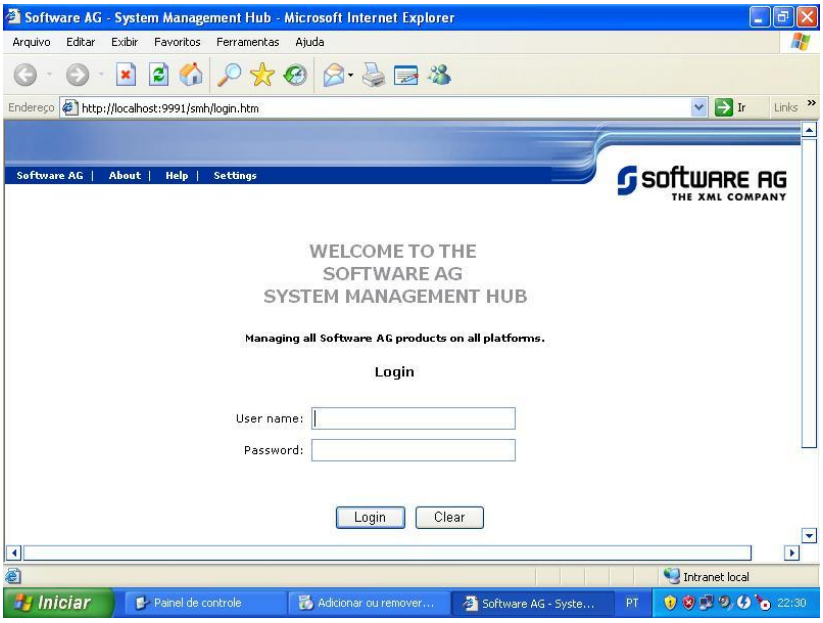

Figura 2.11 – Página inicial do Tamino.

Através desse sistema é possível fazer o gerenciamento completo do *Tamino*, desde a criação de um novo banco de dados até o gerenciamento de consultas.

2.4.3 Apache Xindice

O *Xindice*, antes chamado de dbXML Core, é um banco de dados XML nativo desenvolvido em 1999, na linguagem C/C++. No ano de 2000, o código fonte foi migrado para Java e doado para a *Apache Software Foundation* em dezembro de 2001. (FIGUEIREDO, 2003)

O *Xindice* é um banco de dados gratuito de código aberto que armazena e adiciona índices a documentos XML compactados. Os dados armazenados são fornecidos pelo servidor a uma aplicação cliente com a vantagem de utilizar pouco processamento no lado servidor. Além disso, o *Xindice* oferece suporte a funcionalidades particulares de dados XML que não podem ser facilmente reproduzidas em bancos de dados relacionais.

O objetivo principal do *Xindice* é armazenar e gerenciar um grande número de documentos XML. Tal objetivo é alcançado através do armazenamento de documentos XML comprimidos em coleções de documentos.

2.4.3.1 Arquitetura

A organização dos documentos armazenados no *Xindice* é feita através de coleções hierárquicas. Em cada coleção está contido um determinado número de documentos XML além de suas próprias coleções, gerando assim uma hierarquia. (MENDES)

A coleção raiz do *Xindice*, também chamada de base de dados, é a única com características especiais, pois não possui coleção-pai nem documentos XML e é criada na primeira execução do *Xindice*.

Fisicamente, em cada diretório de uma coleção, existe pelo menos um arquivo binário com todos os dados dos documentos XML armazenados naquela coleção. O nome deste arquivo é formado pelo nome da coleção onde se encontra os documentos XML e pela extensão ".tbl". No diretório */db/teste*, por exemplo, existe um arquivo chamado *teste.tbl*. Esse arquivo é o modelo fundamental de armazenamento físico proprietário do *Xindice*.

Na versão 1.0 do *Xindice*, há suporte para a maioria das características de um NXD, porém, essa versão não possui transações nem controle de concorrência.

O gerenciamento de tarefas do *Xindice* se dá através de linhas de comando e para fazer consultas ao *Xindice*, é utilizada a linguagem de consulta XPath, que será abordada na seção 2.3.

2.4.3.2 Armazenamento

O *Xindice* utiliza a estrutura de armazenamento de uma árvore B, com arquivos paginados. Com isso, há um aumento de desempenho, disponibilizando um fácil mecanismo de consultas e manipulação dos dados armazenados.

Ainda a respeito do armazenamento, o *Xindice* tem a capacidade de armazenar um grande número de pequenos documentos XML de até 50Kb. É possível armazenar documentos maiores, porém a Apache não aconselha esse tipo de armazenamento. O tamanho máximo de um documento XML que o *Xindice* pode ter em uma coleção é de 5Mb. (FIGUEIREDO, 2003)

Durante sua execução, uma instância do *Xindice* armazena alguns dados em objetos Java dentro de uma JVM. Dentre esses dados, os mais importantes são: representação da hierarquia de coleções, estado da conexão com o cliente e dados em *cache*. Além disso, o *Xindice* precisa acessar arquivos do disco contendo dados em formato XML, armazenados em uma hierarquia de diretórios, começando em um diretório chamado *raiz do banco de dados*.

O *Xindice* pode funcionar de duas maneiras: modo embutido e modo servidor. No modo embutido, uma aplicação Java completa faz a configuração de uma instância do *Xindice* em sua própria JVM e apenas tal aplicação está habilitada a acessar e manipular os dados armazenados no *Xindice*. No modo servidor, o *Xindice* é executado como uma aplicação web padrão JEE, podendo estar dentro de um container de aplicações web, como o Apache Tomcat. Nesse modo, vários clientes podem acessar a base de dados XML de diferentes máquinas virtuais, possivelmente em outro computador, com o auxílio do protocolo de chamada XML-RPC, que é um padrão XML para chamadas remotas de procedimento sobre o protocolo HTTP.

2.4.3.3 Download e execução

O download do *Xindice* pode ser feito através do site *http://xml.apache.org/* e o link para download é *http://xml.apache.org/xindice/download.cgi*. No site encontram-se também um tutorial para a instalação e a documentação do NXD.

Não há um instalador para Windows, portanto o usuário deve ter um pouco de paciência para fazer a instalação do *Xindice*. Ele funciona nos sistemas operacionais Windows

e Linux. Para rodar o *Xindice* é necessário um processador *Java Server Pages* (JSP) instalados no sistema, pois o *Xindice* não é autônomo.

Os comandos do *Xindice* são executados através do terminal de comandos do Windows. A figura 2.12 uma interface web para o *Xindice*.

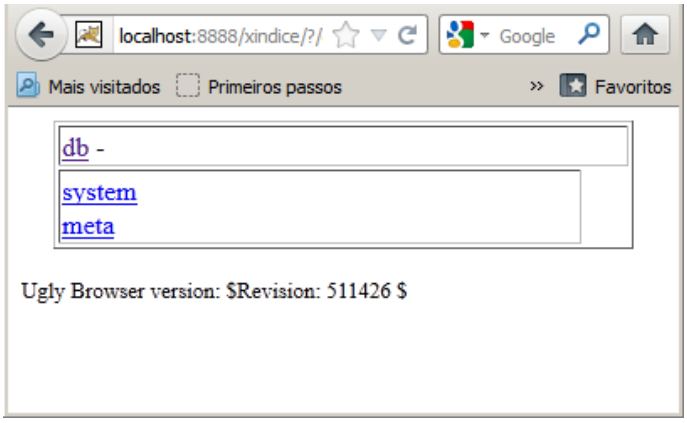

Figura 2.12 - Lista de coleções da base de dados após a instalação.

Os dados presentes na base de dados, bem como as coleções e os documentos XML armazenados no *Xindice*, podem ser visualizados através de um navegador web pelo endereço *http://localhost:8888/xindice*.

2.4.4 Sedna XML DBMS

O *Sedna* é um NXD de licença gratuita e de código aberto desenvolvido pela *Management Of Data & Information Systems (MODIS)* do *Institute for System Programming of the Russian Academy of Sciences*. Atualmente o *Sedna* está na versão 3.5 e pode ser instalado nos sistemas operacionais Windows, Linux, Mac OS, FreeBSD e Solaris. (Sedna, 2012)

Este NXD foi desenvolvido nas linguagens Scheme e C++. A equipe da MODIS decidiu não adotar nenhum SGBD como base de desenvolvimento. O tempo de desenvolvimento e esforço foram muito maiores, mas isso deu aos desenvolvedores maior liberdade na tomada de decisões do projeto. (Sedna, 2012)

2.4.4.1 Arquitetura

A arquitetura do *Sedna* tem os seguintes componentes: *governor, listener, connection, transaction, parser, optimizer, executor*. A seguir a figura 2.13 ilustra a arquitetura do *Sedna*.

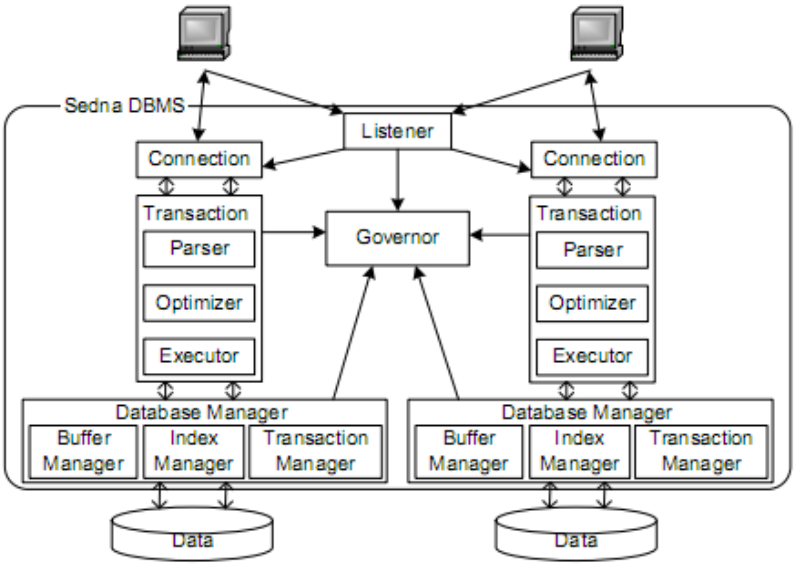

Figura 2.13 - Arquitetura do Sedna. (CUONG, 2006)

O componente *governor* faz o papel de controlador central do sistema. Todos os outros componentes registram-se no *governor*. O *governor* controla todas as bases de dados e transações que estão rodando no sistema. O *listener* é responsável por criar uma instância do componente *connection* para cada cliente e atribuir uma conexão direta entre o cliente e o componente *connection*. O componente *connection* encapsula a sessão do cliente e também cria uma instância do componente *transaction* para cada comando "begin transaction" que o cliente solicitar. O componente *transaction* por sua vez encapsula os componentes de execução de consulta, que são: *parser, optimizer e executor*. O *parser* traduz a consulta em sua representação lógica, o *optimizer* obtém a essa representação lógica e produz o plano de execução de consulta otimizado, que é uma árvore de operações de baixo nível sobre as estruturas de dados físicos. O *executor* é responsável por interpretar o plano de execução (CUONG, 2006).

Cada instância do *database manager* encapsula um único banco de dados e consiste em serviços de gerenciamento de banco de dados, como o *index manager* que se mantém informado dos índices construídos na base de dados, o *buffer manager* que é responsável pela

interação entre disco e memória principal, e o *transaction manager* que oferece recursos de controle de concorrência.

Como a implementação do *Sedna* foi baseada na linguagem de consulta XQuery 1.0, este NXD cobre todas as funcionalidades da biblioteca XQuery e do XQuery Core. A linguagem de consulta XQuery será abordada na seção 2.3.

2.4.4.2 Armazenamento

Para a organização dos dados do *Sedna* foram tomadas duas decisões para acelerar o processamento das consultas. Primeiro, ponteiros diretos são usados para representar o relacionamento entre os nós de um documento XML. Segundo, foi desenvolvido um esquema descritivo que conduz a um armazenamento estratégico. Esse armazenamento consiste em aglomerar nós de um documento XML de acordo com a posição dos nós no esquema descritivo do documento.

Em contraste com o esquema prescritivo que é conhecido de antemão e geralmente especificado no DTD e no XML Schema, o esquema descritivo é dinamicamente gerado a partir dos dados e apresenta uma estrutura concisa e precisa desses dados. Formalmente falando, todo caminho do documento tem exatamente um caminho no esquema descritivo, e todo caminho no esquema descritivo é um caminho do documento.

Como diz na definição, o esquema descritivo é sempre uma árvore. Utilizando o esquema descritivo ao invés do prescritivo têm-se as seguintes vantagens: o esquema descritivo é mais preciso que o prescritivo e também permite aplicar esta estratégia de armazenamento de documentos XML que o prescritivo não permite.

A figura 2.14 ilustra de uma forma geral a organização de armazenamento do *Sedna*. (Sedna, 2012)

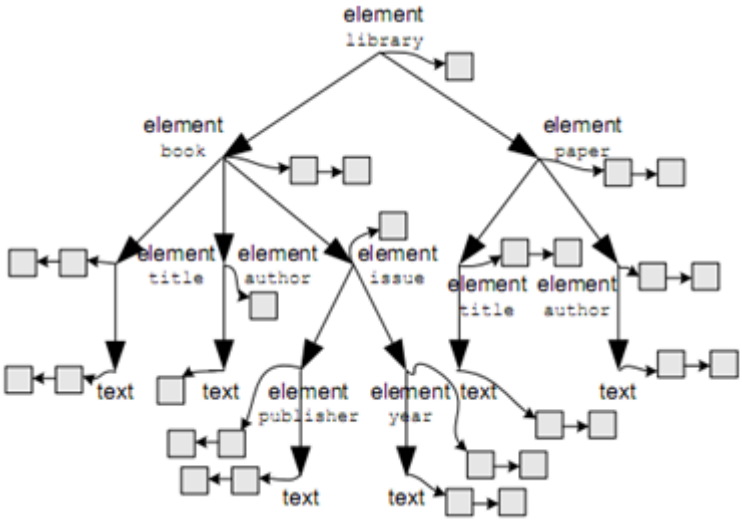

Figura 2.14 - Organização dos dados no Sedna. (CUONG, 2006)

O componente central é o esquema descritivo que é apresentado como uma árvore de nós esquema. Cada nó esquema é rotulado com um tipo de nome de um nó XML (e.g. *element*, *attribute*, *text*, etc.) e tem um ponteiro para blocos de dados em que os nós correspondentes para o nó esquema são armazenados. (CUONG, 2006)

2.4.4.3 Download e execução

O download do *Sedna* pode ser feito através do site *http://www.sedna.org/*. O link para download está na página principal. No site encontram-se também um tutorial para a instalação e documentação bastante completa para ajudar nos primeiros passos na utilização do *Sedna*.

Para inicializar os serviços do *Sedna* é necessário executar o componente principal do servidor que é o *governor*. Como citado antes, o *governor* é o controle central do sistema, por isso deve-se executar o *governor* antes de qualquer coisa. Todas as tarefas executadas no *Sedna* são feitas através do terminal do Windows, por linha de comando.

2.4.5 SQL/XML-IMDB

O *SQL/XML-IMDB* é um banco de dados híbrido de licença comercial desenvolvido pela QuiLogic. Esse banco de dados tem a possibilidade de executar consultas em SQL e XML (QuiLogic, 2012), porém como este trabalho trata da avaliação de bancos de dados

XML Nativo, serão estudadas apenas as funções referentes ao processamento de documentos XML, portanto, será feita referência apenas ao XML deixando de lado o SQL.

O *SQL/XML-IMDB* é uma base de dados representada em XML que foi projetada com o objetivo de unir informações estruturadas e não estruturadas de bases de dados relacionais, documentos XML e diretórios de arquivos normais (MARQUES, 2002). Com esse NXD é possível, de forma fácil, acessar dados relacionais e fazer pesquisa de dados ao longo de documentos XML.

O *SQL/XML-IMDB* é um NXD *in-memory-database*, ou seja, o processamento e gerenciamento dos dados XML são feitos com os dados direto na memória. Com isso o tempo de resposta para consultar dados é muito mais rápido que um banco de dados que possui apenas armazenamento direto em disco.

2.4.5.1 Arquitetura e armazenamento

O *SQL/XML-IMDB* armazena documentos XML na sua estrutura nativa, ou seja, os documentos armazenados permanecem na sua forma normal. Quando os documentos são armazenados ou acessados não há conversão de estrutura.

O *SQL/XML-IMDB* organiza e armazena seus dados em tabelas, uma "tabela" no armazenamento de dados XML é o que a maioria dos outros NXDs se refere a uma coleção, com um documento XML por linha. As tabelas podem ser criadas localmente a um processo particular ou compartilhada entre processos e utilizar compressão para minimizar o uso de memória.

Tanto o armazenamento relacional quanto o XML são indexados com árvores *Ternary search tree*. Outros modos de indexação, apenas para documentos XML, são os mecanismos "*Reverse-Lookup*" e "*Token-Segment-Build-Up*" (BOURRET, 2010). A figura 2.15 mostra como é a arquitetura deste NXD.

O armazenamento de dados nesse NXD inclui tecnologia de compressão de dados para diminuir o uso de memória. Cadeias de caracteres são sempre armazenadas em vetores ou em matrizes de bytes com comprimento variável e booleanos são armazenados em apenas um bit. O *optimizer* é responsável por reotimizar os planos durante a execução baseando-se na informação atual de tamanho dos dados intermediários.

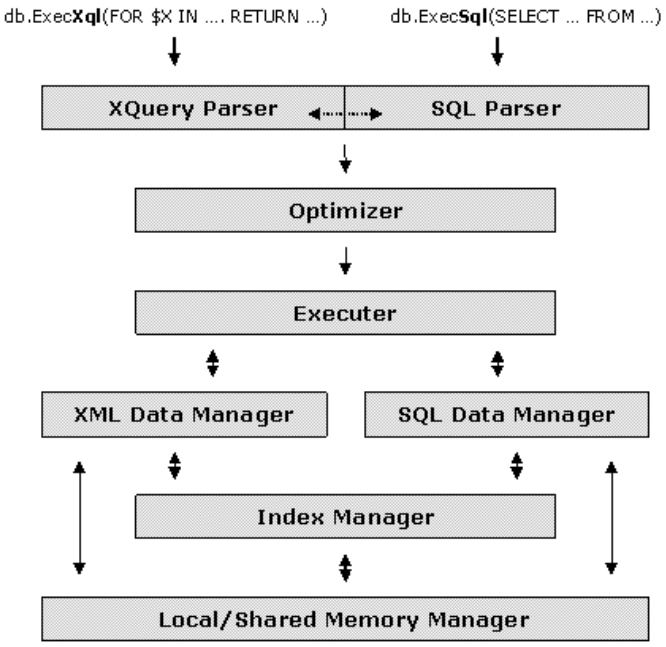

Figura 2.15 - Arquitetura do SQL/XML-IMDB. (QuiLogic, 2012)

O *SLQ/XML-IMDB* oferece suporte tanto a consultas em uma base de dados relacional quanto a documentos XML. Esse NXD possui suporte a um significante subconjunto de funções SQL92 e também possui suporte à linguagem de consulta em NXDs chamada XQuery. Como este trabalho é sobre NXDs, a linguagem estudada será apenas a XQuery, que é mostrada na seção 2.3 deste capítulo.

2.4.5.2 Download e execução

O download do *SQL/XML-IMDB* pode ser feito através do site http://www.quilogic.cc/default.htm.

O link para download é o http://www.quilogic.cc/download.htm. Nos arquivos de instalação do *SQL/XML-IMDB* vêm alguns exemplos de execução desse NXD. Cada exemplo está em uma linguagem de programação diferente, e para executar cada um deles são necessários requisitos especiais de cada linguagem. As linguagens de programação são Delphi, C++, Visual Basic, ASP e ASP.NET.

A execução desse NXD não foi possível, pois os códigos disponibilizados necessitavam de bibliotecas não inseridas no download. A documentação do banco de dados também não resolveu os problemas encontrados para executar o *SQL/XML-IMDB*.

2.5 LINGUAGENS DE CONSULTA A NXDS

Nesta seção será feita uma análise mais detalhada de cada linguagem de consulta de um banco de dados XML nativo.

2.5.1 XPath

XPath é uma linguagem de consulta de documentos XML recomendada pela W3C que permite ao usuário construir expressões que percorrem e processam um documento XML de modo parecido a uma expressão regular.

O modelo de uma expressão XPath é bastante semelhante a um caminho de sistemas de arquivos, por exemplo “C:\Program Files\...”. Essa linguagem é utilizada para selecionar nós ou conjuntos de nós de um documento XML (W3Schools, 2012).

Atualmente, a XPath possui mais de cem funções embutidas e através delas é possível trabalhar com cadeias de caracteres, valores numéricos, comparação de data e tempo, nós de árvores, manipulação de sequência, valores Booleanos, entre outros.

A XPath trata os documentos XML da mesma maneira que o DOM trata, ou seja, como uma árvore de nós. Na XPath existem sete tipos de nós, que são: *elemento*, *atributo*, *texto*, *namespace*, *instrução de processamento*, *comentário* e *nós de documento*. A figura 2.16 mostra um documento XML que será analisado a seguir.

```
<?xml version="1.0" encoding="UTF-8"?>
<dblp>
   <article key="journals/pvldb/FabbriL11" mdate="2011-10-18">
      <author>Daniel Fabbri</author>
      <author>Kristen LeFevre</author>
      <title>Explanation-Based Auditing.</title>
      <pages>1-12</pages>
      <year>2011</year>
      <volume>5</volume>
      <journal>PVLDB</journal>
      <number>1</number>
      <ee>http://www.vldb.org/pvldb/vol5/p001_danielfabbri_vldb2012.pdf</ee>
      <url>db/journals/pvldb/pvldb5.html#FabbriL11</url>
   </article>
</dblp>
```

Figura 2.16 - Documento *FabbriL11.xml.(DBLP, 2012)*

Avaliando a figura acima, o nó <*dblp*> é o nó elemento raiz, <*article*> é um nó elemento e *key=”journals/pvldb/FabbriL11”* é um nó atributo.

Existem também os valores atômicos e os itens. Valores atômicos são nós que não possuem nem pais nem filhos. Por exemplo, *Daniel Fabbri* é um valor atômico. Um item pode ser um valor atômico ou um nó.

O relacionamento entre os nós é dado da mesma forma que em DOM, com dois relacionamentos a mais. Na XPath o tipos de relacionamentos são: *pais, filhos, irmãos, ancestrais e descendentes*. Os relacionamentos *pais*, *filhos* e *irmãos* da XPath seguem o mesmo padrão do DOM.

Para explicar os *ancestrais* pode-se utilizar novamente a figura 2.16. Tomando o nó elemento <*title*> como exemplo, os seus *ancestrais* são os elementos <*article*> e <*dblp*>. No caso dos *descendentes*, seria o contrário. Tomando o nó elemento raiz <*dblp*>, os seus descendentes são os nós elementos <*article*>, <*author*>, <*title*>, etc. A consulta "*/dblp/article/title*" é um exemplo de consulta XPath. Executado esta consulta no documento XML da figura 2.16, retorna o nó elemento <*title*> com todo seu conteúdo. Este nó elemento está localizado dentro do nó elemento <*article*>, que por sua vez é filho do nó raiz <*dblp*>. O valor do retorno é "<*title*>*Explanation-Based Auditing*</*title*>".

Outro exemplo de consulta XPath é "*//title/text()[../../author=”Daniel Fabbri”]*". Nesta segunda consulta, o valor retornado é o valor de qualquer nó elemento "<*title*>" que esteja em qualquer parte do documento XML. Porém este nó elemento deve ter um nó elemento irmão chamado "<*author*>" com o valor igual a "*Daniel Fabbri*".

2.5.2 XQuery

A linguagem XQuery é uma linguagem de consultas para arquivos XML, construída em expressões XPath. Pode-se dizer que a XQuery é para o XML o que o SQL é para as tabelas de um banco de dados (W3Schools, 2012), sendo assim ela é muito mais complexa que a XPath.

A XQuery foi desenvolvida para consultar dados XML. Alguns exemplos de uso da XQuery são: extração de informações para usar em um servidor web; gerar relatórios resumidos; transformar dados XML para XHTML; pesquisa em documentos web para extração de informações relevantes; entre outros.

A seguir são mostrados alguns exemplos de utilização da XQuery.

Utilizando a XQuery é possível extrair dados de um documento XML. Mas primeiro é necessário abrir este documento. A função *doc()* é utilizada para isso. Para fazer uso dela

deve-se passar o nome do arquivo como parâmetro para esta função. Exemplo: *doc("FabbriL11.xml")*.

Da mesma maneira da XPath, a XQuery utiliza expressões de caminho para navegar através dos elementos em um arquivo XML. Como exemplo tem-se o a expressão de caminho *doc("FabbriL11.xml)/dblp/article/title* que irá selecionar todos os títulos do arquivo *FabbriL11.xml*.

Detalhando a consulta acima, o comando */dblp* fará com que seja selecionado o elemento *dblp*. O comando /*article* selecionará todos os elementos *article* dentro do elemento *dblp*. Por fim, o comando */title* selecionará todos os elementos *title* dentro de cada elemento *article*.

A XQuery permite também consultar arquivos XML utilizando expressões FLOWR. FLWOR é um acrônimo para "For, Let, Where, Order by, Return".

Para os usuários que possuem algum conhecimento de SQL, fica mais fácil de entender como utilizar FLWOR.

Utilizando ainda o documento *FabbriL11.xml* da figura 2.16, tem-se a expressão FLWOR da figura 2.19.

```
for $x in doc("FabbriL11.xml")/dblp/article[author="Daniel Fabbri"]
order by $x/title
return $x/title
```

Figura 2.17 - Expressão FLWOR.

Nesta expressão a cláusula *for* seleciona e armazena todos os elementos *article*, que estão dentro do elemento *dblp*, dentro da variável *$x*. A cláusula *where* faz um refinamento da consulta selecionando apenas os elementos *article* com o valor do elemento *author* igual a "Daniel Fabbri". A cláusula *order by* faz a ordenação do resultado retornado pelo elemento *title*. E por fim, a cláusula *return* especifica o que deverá ser retornado. O valor retornado é "*<title>Explanation-Based Auditing</title>*".

No retorno da consulta XQuery, pode-se observar que todo o nó elemento "*<title>*" foi retornado, caso a especificação do retorno fosse "*$x/title/text()*" o valor retornado seria apenas o texto do nó elemento, ou seja, "*Explanation-Based Auditing*".

2.5.3 X-Query

X-Query é uma das linguagens de consulta que podem ser utilizadas no NXD *Tamino*. Ela é baseada na XPath. Além da maioria das habilidades da XPath, a X-Query oferece recuperação de texto e expressões de ordenação do conteúdo retornado. A X-Query possui suporte a pesquisas full-text e também permite que o usuário faça extensões da própria linguagem de consulta utilizando as extensões do servidor *Tamino*, além de poder recuperar informações de esquema para um doctype XML.

Apesar de a X-Query ser baseada nas especificações da XPath, nem todas as funções da XPath estão disponíveis para a X-Query, porém, essas deficiências podem ser implementadas através das extensões do servidor *Tamino*. Algumas destas funções são: *concat*, *contains*, *starts-with*, *string-length*, e outras mais. A X-Query também não possui suporte para variáveis (X-Query, 2002).

Esta seção dispensa exemplos, pois podem ser usados os exemplos da seção 2.5.1. A sintaxe de consultas X-Query é idêntica a da XPath.

2.6 COMPARATIVO ENTRE OS NXDS

Nesta seção encontra-se uma tabela comparativa dos principais itens sobre cada NXD. Os itens comparados são com relação ao sistema operacional que roda o NXD, algumas citações sobre a arquitetura, armazenamento, como executar, a linguagem utilizada para consultas e alguns requisitos necessário para o NXD funcionar.

	eXist	**Tamino**	**Xindice**	**Sedna**	**SQL/XML-IMDB**
Sistemas Operacionais	Linux, Windows e Mac OS.	Windows Server 2000, Windows Server 2003 e Windows XP Professional	Windows e Linux.	Windows, Linux, Mac OS, FreeBSD e Solaris.	Windows XP.
Arquitetura	Baseada em árvore B+. Organizado em	Desenvolvido sobre o banco de dados	Baseada em coleções hierárquicas.	Arquitetura baseada em um gerente	A organização dos dados é feita em

	coleções de dados XML.	ADABAS. Suporta os métodos HTTP *GET, POST, DELETE, HEAD.*		de atividades central. Possui gerente de transações.	"tabelas" ou coleções.
Armazenamento	Em árvore B+. Arquivos paginados de acordo com o DOM. Armazenamento sem definição de esquema.	Pode ser interno ou externo. Externo pode ser em bancos relacionais e obriga a definição do esquema XML.	Em árvore B, com arquivos paginados. Armazena pequenos documentos XML de até 50Kb.	Através de ponteiros para representar a hierarquia.	Em árvores *Ternary search tree*. Armazena de forma nativa, sem conversão da estrutura do documento.
Ling. Consulta	XPath e XQuery.	X-Query e XQuery.	XPath.	XPath e XQuery.	Suporte a SQL e XQuery.
Execução	Gerenciado através de um programa cliente em Java. Dados podem ser vistos em um navegador web.	Gerenciado através de interface web ou sistema de interface desktop.	Gerenciado através do terminal do Windows. Dados podem ser vistos em um navegador web.	Gerenciado através do terminal do Windows.	Não foi possível, pois faltam bibliotecas nos códigos disponibilizados no site.
Licença	Gratuito de código aberto.	Comercial.	Gratuito de código aberto.	Gratuito de código aberto.	Comercial.
Pré-requisitos de instalação ou conhecimentos	JDK 1.6, ainda não roda com JDK 7.	*Apache* 2.0 ou superior.	Servidor Tomcat ou Jetty.		Conhecimento em Delphi, C++, Visual Basic, ASP ou ASP.NET

Após a comparação feita na tabela acima, é possível visualizar que a maioria dos NXDs estudados roda em plataforma Windows, e muitos deles possuem um método semelhante de armazenamento dos arquivos, ou seja, através de uma estrutura de dados árvore. Pode-se verificar também que a linguagem de consulta mais utilizada pelos NXDs é a XQuery, que é recomendada pela W3C. Outro item interessante é o modo como o NXD é gerenciado, pois nem todos possuem um programa cliente que faça isso por eles, alguns devem ser gerenciados por linha de comando, tornando o processo menos intuitivo.

No próximo capítulo é descrita a escolha do NXD para o qual foi desenvolvida uma aplicação, que é o objetivo deste trabalho.

3 PROPOSTA DO TRABALHO

Este capítulo apresenta a proposta de desenvolvimento de uma ferramenta que utiliza um dos NXDs estudados para efetuar consultas, inserção, alteração e remoção de documentos de sua base de dados.

3.1 PROPOSTA DE DESENVOLVIMENTO

A ferramenta desenvolvida tem como objetivo acessar um NXD e efetuar consultas, inserção, alteração e remoção de documentos XML da base de dados do NXD.

O NXD escolhido para armazenar os documentos XML foi o eXist. O eXist foi totalmente desenvolvido em Java, tornando-se mais fácil a utilização da mesma linguagem para o desenvolvimento da aplicação. Este NXD foi escolhido por além de ser gratuito, utilizar a linguagem de consulta XQuery, que é uma recomendação da W3C, possuir ótima documentação online e por ser o mais popular dentre os NXDs estudados.

A ferramenta possui interface gráfica e foi desenvolvida na linguagem de programação Java. A aplicação comunica-se com o eXist através do protocolo de comunicação HTTP. O desenvolvimento do código e da interface gráfica foram feitos com o NetBeans IDE 7.0 e a interface gráfica utiliza a ferramenta Swing, que pode ser utilizada pela linguagem Java.

Os dados do eXist utilizados para testes da ferramenta foram extraídos do site *The DBLP Computer Science Bibliography*. O site pode ser acessado pelo endereço http://www.informatik.uni-trier.de/~ley/db/. O site disponibiliza para download documentos XML que tratam de artigos publicados na área de Ciência da Computação. Nos documentos XML disponibilizados são informados quais os autores do artigo, o título, volume, entre outros.

A figura 3.1 mostra o modelo do documento XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<dblp>
   <article key="journals/pvldb/GoyalBL11" mdate="2011-10-26">
      <author>Amit Goyal 0002</author>
      <author>Francesco Bonchi</author>
      <author>Laks V. S. Lakshmanan</author>
      <title>A Data-Based Approach to Social Influence Maximization.</title>
      <pages>73-84</pages>
      <year>2011</year>
      <volume>5</volume>
      <journal>PVLDB</journal>
      <number>1</number>
      <ee>http://www.vldb.org/pvldb/vol5/p073_amitgoyal_vldb2012.pdf</ee>
      <url>db/journals/pvldb/pvldb5.html#GoyalBL11</url>
   </article>
</dblp>
```

Figura 3.1 - Documento GoyalBL11.xml, inserido no eXist para testes da aplicação.

Ambos possuem um nó raiz chamado *<dblp>*. Dentro do nó raiz contém o nó elemento *<article> e dentro dele, contêm os nós elemento <author>*, *<title>*, *<pages>*, *<year>*, *<volume>*, *<journal>*, *<number>*, *<ee>* e *<url>*, sempre dispostos nesta ordem.

Os documentos devem ser inseridos em uma coleção de documentos XML já existente. Para validar os documentos inseridos é possível utilizar a linguagem XML Schema. O documento "*.xsd*" deve ser inserido através do programa cliente do eXist, disponível no site do NXD, pois a ferramenta não possui essa funcionalidade.

3.2 DIAGRAMA DE CASOS DE USO

Nesta seção é apresentado um diagrama de casos de uso da ferramenta a ser desenvolvida. A figura 3.2 mostra o diagrama.

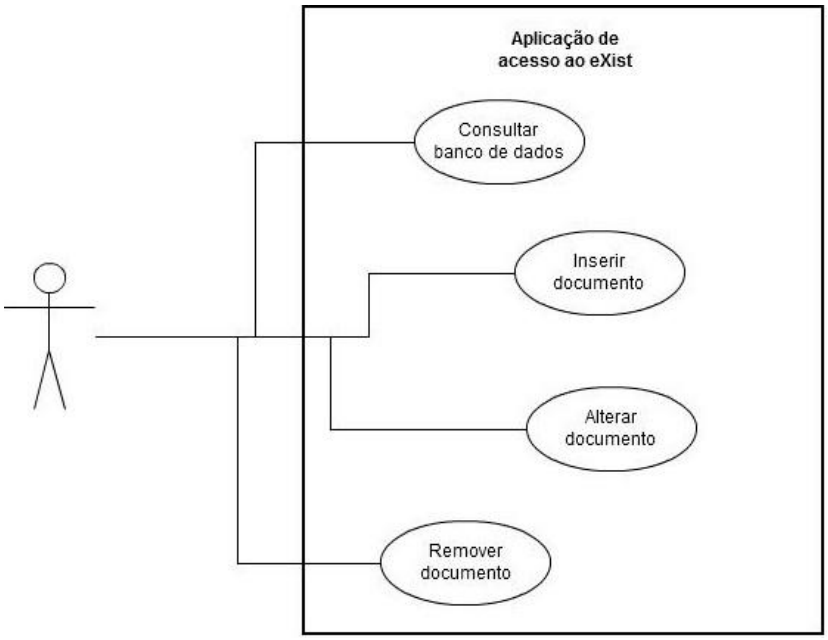

Figura 3.2 - Diagrama de caso de uso da aplicação.

O diagrama de casos de uso apresentado na figura 3.2 é especificado nas subseções a seguir. Para o usuário efetuar qualquer ação descrita abaixo, o mesmo deverá estar logado.

3.2.1 Especificação do caso de uso Consultar banco de dados

Na tela inicial da aplicação, o usuário digitará o nome da coleção de documentos XML a qual ele deseja fazer uma consulta no campo texto com *label* “Coleção”.

O usuário deverá escrever a consulta XQuery no campo de texto com *label* “XQuery” e enviar a consulta para o banco clicando no botão chamado “Enviar consulta” localizado na tela.

Em um cenário normal, a consulta será enviada para a aplicação e em seguida fará uma busca pelas informações solicitadas pelo usuário no NXD, retornando-as e mostrando, caso exista algum conteúdo, o resultado da pesquisa na área de texto com *label* “Resultado da consulta”.

Em um cenário de exceção, caso a consulta contenha erros, será exibido um erro na área de texto com *label* “Resultado da consulta”.

3.2.2 Especificação do caso de uso Inserir documento

Na tela inicial da aplicação, o usuário poderá inserir um documento XML clicando no botão “Inserir Documento XML”. Após clicar no botão, abrirá uma nova tela para o usuário informar a coleção à qual o documento será inserido e selecionar o documento que deseja inserir no NXD, após ter selecionado o documento, o usuário deverá clicar no botão “Enviar”.

Em um cenário normal, a aplicação receberá os dados do documento XML e fará a inserção do mesmo no NXD, fechando a tela de inserção e exibindo uma mensagem de sucesso da inserção em um alerta.

Em um cenário de exceção, caso ocorra algum erro durante o processo de inserção, a tela de inserção será fechada e será exibido um erro em um alerta.

3.2.3 Especificação do caso de uso Alterar documento

Na tela inicial da aplicação, o usuário poderá alterar um documento XML através de uma consulta XQuery Update. Os passos para alterar um documento são os mesmos da seção 3.2.1, exceto que ao invés de uma consulta XQuery normal, o usuário utilizará uma consulta XQuery Update.

Em um cenário normal, os dados serão enviados para a aplicação, que modificará o documento XML informado previamente. Após as modificações terem sido feitas, a tela de alteração de documentos será fechada e será exibida uma mensagem de sucesso da modificação em um alerta.

Em um cenário de exceção, caso ocorra algum erro durante o processo de modificação do documento, a tela de alteração de documentos será fechada e será exibida uma mensagem de erro em um alerta.

3.2.4 Especificação do caso de uso Remover documento

Na tela inicial da aplicação, o usuário poderá remover um documento XML clicando no botão “Remover Documento XML”. Após clicar no botão, abrirá uma nova tela para o usuário digitar o nome do documento XML no campo texto com *label* “Documento” e o nome da coleção, onde se encontra o documento, no campo texto com *label* “Coleção”. Após preencher estes campos, o usuário deverá clicar no botão “Remover”.

Em um cenário normal, as informações serão enviadas para a aplicação e após o documento ter sido removido, a tela de remoção será fechada e uma mensagem de sucesso da remoção será exibida em um alerta.

Em um cenário de exceção, caso ocorra algum erro durante o processo de remoção do documento, a tela de remoção de documentos será fechada e será exibida uma mensagem de erro em um alerta.

No próximo capítulo será abordado o desenvolvimento da aplicação proposta para este trabalho.

4 DESENVOLVIMENTO DA APLICAÇÃO

Neste capítulo será discutida a implementação da ferramenta de gerenciamento do NXD eXist. Ao final, será feita uma análise do software desenvolvido.

O desenvolvimento da aplicação foi feito na linguagem Java no ambiente de desenvolvimento integrado (IDE – *Integrated Development Environment*) NetBeans 7.0 (NETBEANS, 2012). Através deste IDE torna-se prático e fácil programar em Java. Além disso, ele possibilita a criação de interfaces gráficas utilizando as ferramentas do Swing. O NetBeans também possui bibliotecas que dão suporte a manipulação de documentos XML.

O diagrama de classes apresentado na figura 4.1 mostra a organização da aplicação desenvolvida. Os atributos das classes foram suprimidos para simplificar o diagrama.

4.1 CLASSES IMPLEMENTADAS

Nesta subseção é apresentado o diagrama de classes da aplicação desenvolvida.

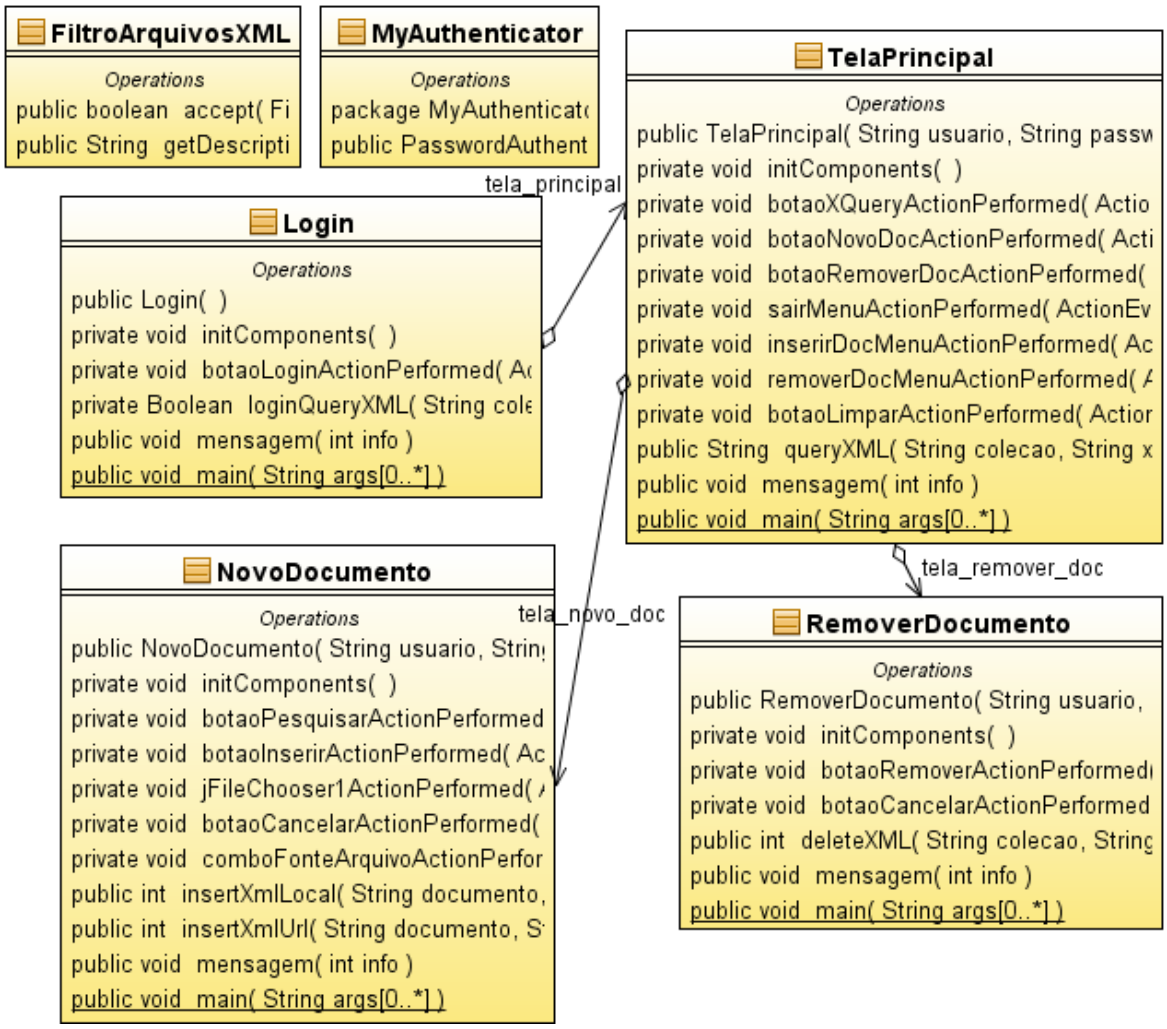

Figura 4.1 - Diagrama de classes da aplicação.

Nas subseções a seguir são apresentadas as classes do diagrama de classes e suas principais funcionalidades.

4.1.1 Classe Login

Esta é a classe que dá a partida da aplicação. Ela implementa a interface da tela de login e o método para validação de login. O método *loginXqueryXML* recebe a coleção onde está armazenado o documento com as informações dos usuários e uma consulta para verificar a validade do login. No eXist, é utilizado o documento "*users.xml*", armazenado na coleção "*system*".

A verificação para saber se os dados informados pelo usuário (nome de usuário e senha) estão corretos é feita através do atributo "*hits*" retornado dentro da tag padrão do eXist "<*result*>" (figura 4.4), no retorno da consulta ao banco. Se o valor do atributo "*hits*" for

igual a 1 (um), é criada uma instância da tela principal da aplicação e a tela de login é encerrada.

A figura 4.2 mostra a consulta XQuery e a chamada do método *loginXqueryXML* para validar o login no NXD.

```
String xquery_login = "for $x in doc(\"users.xml\")/auth/users/user"
        + "[@name=\"" + user + "\" and @password=\"" + password_md5 + "\"]"
        + "return $x";
retorno_login = loginQueryXML("system", xquery_login);
```

Figura 4.2 - Consulta de login.

Na consulta são inseridos o nome do usuário e a senha. A senha deve ser codificada com o método *md5* da classe *MessageDigester* do eXist. Após a consulta efetuada, é retornado verdadeiro ou falso para informar a validade da tentativa de login.

4.1.2 Classe TelaPrincipal

Esta classe implementa a interface da tela principal da aplicação e o método para o envio das consultas XQuery ao NXD.

As consultas, digitadas no campo XQuery da tela principal, são enviadas através de uma conexão HTTP, que utiliza o método de requisição GET. A figura 4.3 mostra um trecho do código do método *queryXml*, que é utilizado para fazer consultas ao NXD.

```
public String queryXml(String colecao, String xquery) {
    try {
        ...
        // ANÁLISE DA String PELO OBJETO SAXBuilder E ANÁLISE DO RETORNO DO NXD
        doc = (org.jdom2.Document) parser.build(new StringReader(resultado));
        element = doc.getRootElement();
        attributes = element.getAttributes();
        hits = attributes.get(ATTR_HITS).getIntValue();
        count = attributes.get(ATTR_COUNT).getValue();
        if (hits == 0) {
            resultado = "A busca não retornou nenhum resultado para a pesquisa.";
        } else if (hits > 0) {
            resultado = str.toString();
            numRegsLabel.setText(count);
        }
        return resultado;
    } catch (Exception e) {
        ...
    }
}
```

Figura 4.3 - Trecho do código do método *queryXml*.

O método *queryXml* recebe dois parâmetros, o nome da coleção e a consulta XQuery. Após o envio da consulta para o NXD, se houver dados de retorno, estes dados são passados para uma variável do tipo *String* e analisados por um parser SAXBuilder, que irá gerar um documento JDOM. A partir deste documento gerado, é possível manipular os dados XML retornados pela consulta XQuery.

Em toda consulta XQuery ao NXD, é retornado o cabeçalho mostrado na figura 4.4. Através dele é possível saber se a consulta retornou algum dado e quantos registros foram retornados.

```
<result xmlns:exist="http://exist.sourceforge.net/NS/exist" hits="8" start="1" count="1">
```

Figura 4.4 - Cabeçalho de uma consulta XQuery.

O atributo "*xmlns*" informa o namespace utilizado, neste caso o do NXD eXist. Os atributos "*hits*" e "*count*" informam se algum dado foi retornado e o número de registros encontrados. Ambos os atributos, "*hits*" e "*count*" fornecem a mesma informação. No exemplo mostrado na figura 4.4, a consulta executada encontrou oito registros.

Na interface da tela principal da aplicação também estão os botões para inserir e remover documentos. O mesmos comandos também podem ser acessados pelo menu "*Arquivos*" ou através de atalhos do teclado. Os atalhos são especificados ao lado do nome de cada opção do menu.

4.1.3 Classe NovoDocumento

Esta classe implementa a interface da tela de inserção de documentos XML e os métodos utilizados para tal função. Os documentos a serem inseridos podem ser obtidos em uma fonte local, ou seja, em um diretório de arquivos do sistema operacional e também através de uma URL.

Na inserção de um documento XML armazenado localmente, o método de inserção utilizado *é o insertXmlLocal*, o qual é exibido um trecho do código na figura 4.5.

Como mostra na figura, o método recebe o caminho completo do documento XML e o nome da coleção onde o documento será armazenado. Antes de qualquer ação, é definido um autenticador padrão, através da classe *Authenticator* do Java, que define um usuário e uma senha para a conexão HTTP. Este passo é necessário para fazer uma alteração no NXD. O usuário e a senha utilizados neste processo são os mesmos informados na tela de *login*.

```
public int insertXmlLocal(String documento, String colecao) {
    // AUTENTICAÇÃO HTTP
    Authenticator.setDefault(new MyAuthenticator(user, pass));
    ...
    try {
        URL url = new URL("http://localhost:8080/exist/rest/db/"
            + colecao + "/" + docName);

        HttpURLConnection connect = (HttpURLConnection) url.openConnection();
        connect.setRequestMethod("PUT");
        connect.setDoOutput(true);
        connect.setRequestProperty("ContentType", "text/xml");

        OutputStream os = connect.getOutputStream();
        InputStream is = new FileInputStream(file);
        byte[] buf = new byte[1024];
        int c;
        while ((c = is.read(buf)) > -1) {
            os.write(buf, 0, c);
        }
        ...
    } catch (Exception e) {
        ...
    }
}
```

Figura 4.5 - Trecho de código do método *insertXmlLocal*.

Após isso, é feita a leitura do documento e através do método de requisição *PUT*, do protocolo HTTP, o documento é armazenado no NXD. Depois dos comandos de inserção, o método *insertXmlLocal* retorna um valor inteiro, que define se o documento foi armazenado com sucesso no NXD ou não.

Alternativamente, o usuário pode inserir um documento XML que se encontra armazenado na web. Para isso é utilizado o método *insertXmlUrl*. Este método recebe a URL onde o documento XML está armazenado, o nome da coleção onde ele será armazenado no NXD e um nome para o documento do NXD. Ambos os métodos de inserção são semelhantes, o que muda de um para o outro é o modo de leitura do documento XML e o número de parâmetros.

4.1.4 Classe RemoverDocumento

Esta classe implementa a interface da tela de remoção de documentos XML e o método responsável por isso.

Para remover um documento XML do NXD, é utilizado o método *deleteXml*, que é mostrado um trecho de seu código na figura 4.6.

```
public int deleteXml(String colecao, String docName) {
    // AUTENTICAÇÃO HTTP
    Authenticator.setDefault(new MyAuthenticator(user, pass));

    try {
        URL u = new URL("http://localhost:8080/exist/rest/db/"
                + colecao + "/" + docName);
        HttpURLConnection conn = (HttpURLConnection) u.openConnection();
        conn.setRequestMethod("DELETE");
        conn.connect();
        ...
    } catch (Exception e) {
        ...
    }
}
```

Figura 4.6 - Trecho de código do método *deleteXml*.

O método *deleteXml* recebe o nome da coleção onde o documento está armazenado no NXD e o nome do documento. Assim como os métodos de inserção de documentos XML, o método de remoção também precisa utilizar a classe Authenticator do Java para definir um usuário e senha para a conexão com o NXD. O método *deleteXml* faz uso do método de requisição DELETE, do protocolo de comunicação HTTP, para a remoção de documentos XML do NXD, e retorna um número inteiro informando se o documento foi removido com sucesso ou não.

4.2 UTILIZAÇÃO DA APLICAÇÃO

A aplicação desenvolvida permite ao usuário armazenar documentos XML no NXD eXist, bem como consultar, editar ou remover estes documentos.

As figuras 4.7 e 4.8 mostram um trecho dos documentos XML que serão utilizados na demonstração da aplicação.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<authors>
    <author id="1">
        <name>Graham Cormode</name>
        <institution>Test</institution>
    </author>
    <author id="2">
        <name>Justin Thaler</name>
        <institution>Test</institution>
    </author>
    ...
    <author id="37">
        <name>Andreas Lüdtke</name>
        <institution>Test</institution>
    </author>
    <author id="38">
        <name>Yll Haxhimusa</name>
        <institution>Test</institution>
    </author>
</authors>
```

Figura 4.7 - Documento "*authors.xml*".

O documento "*authors*.xml" contém as informações dos autores dos artigos e livros mostrados na figura 4.8.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<dblp>
    <article key="journals/pvldb/CormodeTY11" mdate="2011-10-18">
        <author><authorref idref="1"/></author>
        <author><authorref idref="2"/></author>
        <author><authorref idref="3"/></author>
        <title>Verifying Computations with Streaming Interactive Proofs.</title>
        <pages>25-36</pages>
        <year>2011</year>
        <volume>5</volume>
        <journal>PVLDB</journal>
        <number>1</number>
        <ee>http://www.vldb.org/pvldb/vol5/p025_grahamcormode_vldb2012.pdf</ee>
        <url>db/journals/pvldb/pvldb5.html#CormodeTY11</url>
    </article>
    ...
    <book key="books/daglib/0026809" mdate="2011-08-26">
        <author><authorref idref="38"/></author>
        <title>The Structurally Optimal Dual Graph Pyramid...Partitioning.</title>
        <pages>I-XXIII, 1-193</pages>
        <publisher>IOS Press</publisher>
        <year>2007</year>
        <series href="db/series/diski/index.html">DISKI</series>
        <volume>308</volume>
        <isbn>978-1-58603-743-7</isbn>
    </book>
</dblp>
```

Figura 4.8 - Documento "*dblp.xml*".

Os dados destes dois documentos XML, apresentados nas figuras acima, foram condensados para otimizar espaço.

Nos documentos apresentados, foi utilizada a ideia de referências para o possível cruzamento dos dados dos dois documentos. Explicando melhor o que são as referências, no documento "*authors.xml*", pode-se tomar a linha 3 como exemplo. O nó elemento "<*author*>" possui um atributo chamado "*id*", que é o identificador do autor, ou seja, uma chave única para cada autor. No documento "*dblp.xml*", utilizando a linha 4 como exemplo, também existe um nó elemento "<*author*>" que, por sua vez, possui um nó elemento filho chamado "<*authorref*>" com o atributo "*idref*". O atributo "*idref*" é uma referência para o atributo "*id*" do elemento "*author*" do documento "*authors.xml*". Cruzando os dados, chega-se a conclusão de que na linha 4 do documento "*dblp.xml*", o atributo de referência "" se refere ao autor "*Graham Cormode*" do documento "*authors.xml*".

Primeiramente, é necessário adicionar os documentos em alguma coleção do NXD.

4.2.1 Inserção de documentos XML no eXist

Através da tela principal da aplicação, o usuário pode clicar no botão "*Inserir documento XML*" ou acessar esta mesma opção no menu "*Arquivos*". Então será exibida a tela de inserção de documentos, mostrada na figura 4.9.

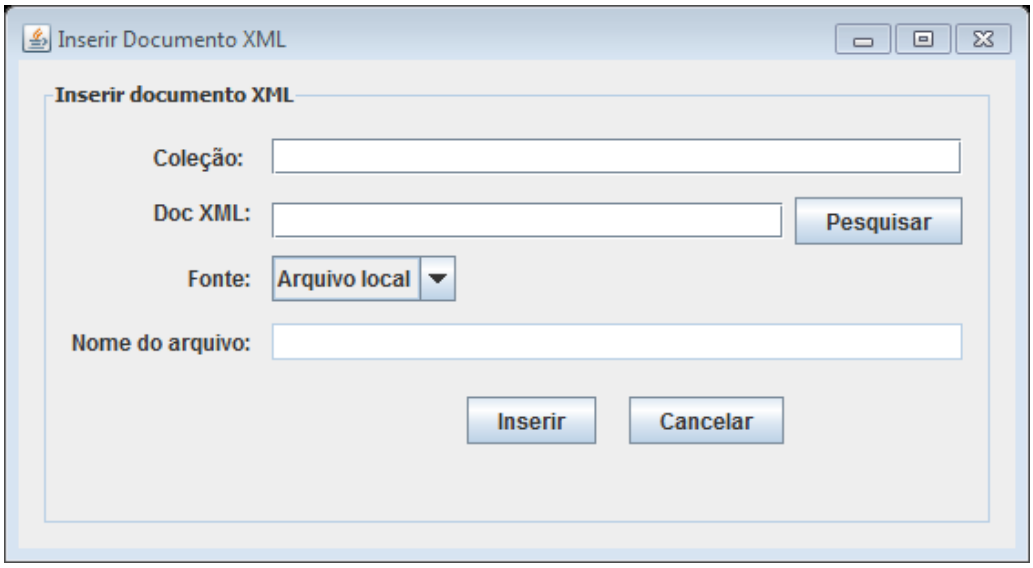

Figura 4.9 - Tela de inserção de documentos XML.

Nesta tela o usuário deverá informar a coleção onde o documento XML será armazenado no NXD. O eXist aceita que o usuário adicione uma coleção dentro de outra coleção, criando assim uma subcoleção. Exemplificando, existem duas coleções, chamadas col1 e col2. A col2 é uma subcoleção da col1, portanto se o usuário deseja inserir um documento dentro da coleção col2, ele deverá informar o caminho inteiro desta coleção, ou seja, “col1/col2”. Após isso, o usuário deve selecionar um arquivo local através do botão “Pesquisar” ou colar a URL, onde o documento XML está armazenado, no campo “Doc XML”. Caso seja escolhida a segunda opção, ou seja, URL, o usuário deverá mudar o campo “Fonte” para “URL” e preencher o campo “Nome do arquivo” de forma que possa identificar futuramente o documento no NXD. É importante saber quais documentos já existem no NXD, pois se o usuário escolher um nome que já existe no NXD, o documento será sobrescrito.

4.2.2 Execução de consultas XQuery no eXist

Após a inserção do documento XML ter sido efetuada com sucesso no NXD, é possível escrever consultas XQuery para buscar os dados armazenados. Para isso o usuário deverá saber em qual coleção o documento XML está armazenado e quais documentos o ele deseja consultar. A figura 4.10 mostra a tela principal da aplicação.

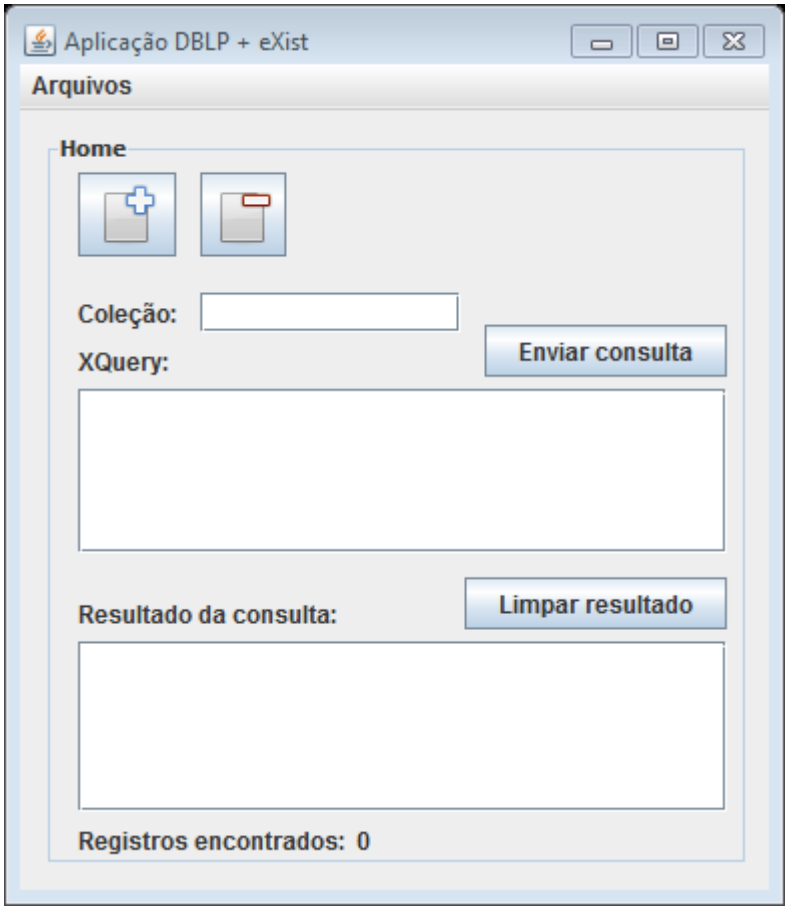

Figura 4.10 - Tela principal da aplicação.

No exemplo a seguir foram utilizados os documentos "*authors.xml*" e "*dblp.xml*", mostrados anteriormente. Eles foram armazenados na coleção "*dblp*". A consulta, para retornar os autores e títulos dos artigos presentes no documento, é mostrada na figura 4.11.

```
1  <articles>
2      {
3      for $a in fn:doc("authors.xml")/authors/author,
4          $b in fn:doc("dblp.xml")/dblp/article[author/authorref/@idref = $a/@id]
5          order by $a/name, $b/title
6      return
7          <author_title>
8              <author>{$a/name/text()}</author>
9                  <title>{$b/title/text()}</title>
10         </author_title>
11     }
12 </articles>
```

Figura 4.11 - Consulta XQuery.

Nesta consulta é feito uma junção dos documentos "*authors.xml*" e "*dblp.xml*". Na linha 3 é definido o caminho de busca no documento "*authors.xml*" e na linha 4 é definido o caminho de busca do documento "*dblp.xml*" e feita a junção dos dois documentos de acordo

com as informações dos autores, presente nos atributos "*id*" e "*idref*". Na linha 5 o resultado da junção é ordenado alfabeticamente pelos nomes dos autores e pelos títulos dos artigos. A partir da linha 6 até a linha 10, é definido como serão formatados e quais os dados do retorno da consulta.

O retorno da consulta é mostrado na figura 4.12. As informações do retorno foram comprimidas para otimizar o espaço.

```
<articles>
    <author_title>
        <author>Adam Marcus 0002</author>
        <title>Human-powered Sorts and Joins.</title>
    </author_title>
    ...
    <author_title>
        <author>Nadathur Satish</author>
        <title>Fast Updates on Read-Optimized Databases Using Multi-Core CPUs.</title>
    </author_title>
</articles>
```

Figura 4.12 - Retorno da consulta XQuery.

Como podemos ver, os dados de ambos os documentos foram unidos através das referências dos autores, presente nos dois documentos.

Outro exemplo de consulta XQuery é a consulta mostrada na figura 4.13.

```
1  for $a in fn:doc("authors.xml")/authors/author,
2        $b in fn:doc("dblp.xml")/dblp/article[author/authorref/@idref = $a/@id]
3  where $b/url/j = "pvldb" and $b/url/v = "5"
4  return
5      <data>
6          {$a/name}
7          {$b/title}
8          {$b/journal}
9          {$b/volume}
10         {$b/url}
11     </data>
```

Figura 4.13 - Consulta XQuery com conteúdo misto.

Nesta consulta é feita a mesma junção da consulta mostrada na figura 4.11. A diferença desta consulta, é que os dados consultados estão em um conteúdo misto. Para melhor entendimento, observe a figura 4.14.

```
1  <dblp>
2      <article key="journals/pvldb/CormodeTY11" mdate="2011-10-18">
3          <author>
4              <authorref idref="1"/>
5          </author>
6          <author>
7              <authorref idref="2"/>
8          </author>
9          <author>
10             <authorref idref="3"/>
11         </author>
12         <title>Verifying Computations with Streaming Interactive Proofs.</title>
13         <pages>25-36</pages>
14         <year>2011</year>
15         <volume>5</volume>
16         <journal>PVLDB</journal>
17         <number>1</number>
18         <ee>http://www.vldb.org/pvldb/vol5/p025_grahamcormode_vldb2012.pdf</ee>
19         <url>db/journals/pvldb/<j>pvldb</j><v>5</v>.html#CormodeTY11</url>
20     </article>
21 </dblp>
```

Figura 4.14 – Documento XML com conteúdo misto.

Esta figura mostra um exemplo de documento XML com conteúdo misto, ou seja, o conteúdo deste documento possui um nó elemento que além de um valor textual, ele também possui outros nós elementos como filhos. Analisando a figura 4.14, na linha 19, o nó elemento "*url*" possui um valor textual mesclado com os nós elementos "*j*" e "*v*", que por sua vez também possuem texto dentro deles.

Voltando a consulta mostrada na figura 4.13, a linha 3 é responsável por buscar o artigo do documento "*dblp.xml*", onde o nó elemento "*j*" seja igual ao valor "*pvldb*" e o nó elemento "*v*" seja igual ao valor "*5*". Então são retornados, dentro do nó elemento "*data*", os seguintes dados: nome do autor, título do artigo, a revista onde foi publicado, o volume e a url. A figura 4.15 mostra o resultado da consulta.

```
<data>
    <name>Graham Cormode</name>
    <title>Verifying Computations with Streaming Interactive Proofs.</title>
    <journal>PVLDB</journal>
    <volume>5</volume>
    <url>db/journals/pvldb/<j>pvldb</j>
        <v>5</v>.html#CormodeTY11</url>
</data>
<data>
    <name>Justin Thaler</name>
    <title>Verifying Computations with Streaming Interactive Proofs.</title>
    <journal>PVLDB</journal>
    <volume>5</volume>
    <url>db/journals/pvldb/<j>pvldb</j>
        <v>5</v>.html#CormodeTY11</url>
</data>
<data>
    <name>Ke Yi</name>
    <title>Verifying Computations with Streaming Interactive Proofs.</title>
    <journal>PVLDB</journal>
    <volume>5</volume>
    <url>db/journals/pvldb/<j>pvldb</j>
        <v>5</v>.html#CormodeTY11</url>
</data>
```

Figura 4.15 - Retorno da consulta ao conteúdo misto.

Apenas como observação, o documento XML com conteúdo misto é uma modificação de um documento XML do DBLP. Esta modificação foi feita apenas para exemplificar uma consulta em um documento com conteúdo misto.

4.2.3 Alteração de documentos XML do eXist

Com os documentos na base de dados, é possível fazer alterações neles sem ser necessário sobrescrever estes documentos com versões atualizadas. Em alguns casos os documentos são muito extensos, e seria mais custoso para o NXD remover e depois reescrever todo o conteúdo de um documento grande para manter seus dados atualizados.

A atualização desses documentos é feita através de uma consulta XQuery Update. A consulta deve ser digitada na tela principal, da mesma forma que uma consulta XQuery normal. A figura 4.16 mostra um exemplo de consulta XQuery Update.

```
1  for $a in fn:doc("authors.xml")/authors/author,
2          $b in fn:doc("dblp.xml")/dblp/
3            article[author/authorref/@idref = $a/@id]
4        where $a/name = "Graham Cormode"
5    return update replace $b/pages/text() with "26-35"
```

Figura 4.16 - Consulta XQuery Update.

Na consulta de atualização apresentada acima, o documento a ser atualizado é o "*dblp.xml*". O artigo a ser atualizado é o que possui o autor "*Graham Cormode*" na lista de autores. Nas linhas de 1 a 3 é feito uma junção dos documentos para que na linha 4 o autor possa ser buscado pelo nome. Na linha 5 é dado o comando para atualizar o conteúdo do elemento "*pages*" do documento "*dblp.xml*", onde o conteúdo deste elemento é substituído pelo texto "*26-35*".

4.2.4 Remoção de documentos XML do eXist

A aplicação também permite ao usuário remover documentos através da tela de remoção. Para acessar a tela de remoção de documentos XML, o usuário pode clicar no botão "*Remover documento XML*" na tela principal, ou acessar o menu "*Arquivos*" e selecionar a mesma opção. A figura 4.17 mostra a tela de remoção de documentos XML.

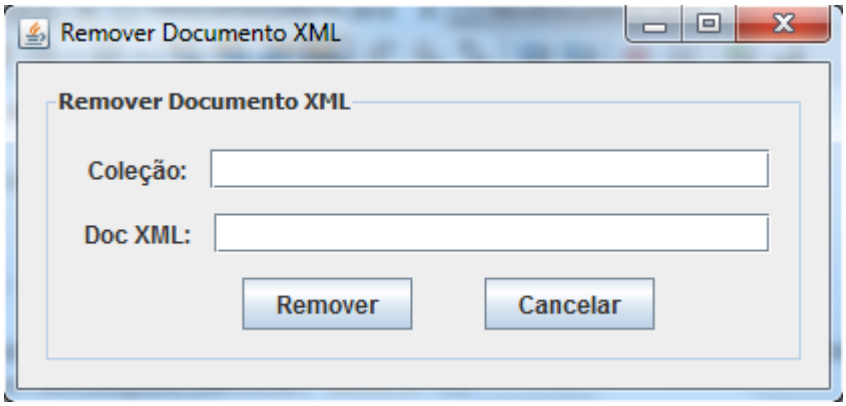

Figura 4.17 - Tela de remoção de documentos XML.

Para remover um documento XML da base de dados do NXD, o usuário deve saber em qual coleção o documento está armazenado e o nome correspondente ao documento armazenado no banco de dados.

4.2.5 Tratamento de erros

Na fase de testes da aplicação foram identificados alguns erros que o usuário poderá cometer. O primeiro deles é a respeito do preenchimento dos campos texto. Todos eles são obrigatórios, exceto os campos inativos, como por exemplo o campo "*Nome*" na tela de inserção de documentos XML. Este campo fica inativo quando a fonte de inserção escolhida é

a local e não URL. Porém quando é selecionada a fonte URL, o campo "*Nome*" se torna obrigatório. Outro possível erro é no caso de uma inserção o usuário informar o caminho errado do documento. Se a aplicação não conseguir ler o documento, será exibido um alerta informando ao usuário que não foi possível abrir o documento. Ainda na inserção de documentos, foram tratados os erros para os seguintes casos: falha na formatação do documento, ausência da extensão "*.xml*" no final do nome do documento, falha na autenticação do usuário (usuário não autorizado) e erro na conexão com o NXD (caso o mesmo não tenha sido inicializado).

Na tela de remoção de documentos foi tratado o erro de documento não encontrado. Caso o usuário informe um documento que não exista no NXD ou na coleção informada, é retornado para o usuário um alerta dizendo que o documento não foi encontrado.

Por último, mas não menos importante, na tela de login, foi feito o tratamento para erro de conexão com o NXD e a sua principal finalidade que é a verificação de login.

4.3 ANÁLISE DA APLICAÇÃO DESENVOLVIDA

Esta seção apresenta uma análise das funcionalidades implementadas na aplicação de acesso ao NXD eXist.

4.3.1 Inserção de documentos XML

Para realizar a inserção de documentos XML no NXD eXist, a aplicação realiza primeiramente a definição de uma autenticação HTTP, configurando um usuário e uma senha, para a inserção do documento XML no NXD através do método HTTP chamado PUT. Em seguida, é feita a verificação do nome do documento, para se certificar de que é um documento XML, ou seja, que o documento possui a extensão "*.xml*". Após esta verificação é feita a tentativa de abertura do documento, caso este processo seja concluído com sucesso, é criada uma conexão HTTP através da URL de acesso ao NXD. Então, é feita a leitura e inserção dos dados do documento XML no NXD.

4.3.2 Consultas e atualização dos documentos XML

Através de uma conexão HTTP, é utilizado o método GET para obter o retorno da consulta XQuery enviada na URL de comunicação com o eXist. Os dados retornados são então processados para dentro de uma variável do tipo *StringBuilder* e futuramente convertidos para dentro de outra variável do tipo String. Posteriormente, esta última string é lida por um *parser SAXBuilder*, tornando possível a manipulação dos dados XML retornados pelo NXD. Após uma verificação, se houver registros no retorno do NXD, eles são enviados para a tela principal, onde são exibidos para o usuário.

A atualização dos documentos é feita pelo mesmo método de consultas, a diferença é que nenhum dado de retorno é tratado.

4.3.3 Remoção de documentos XML

Da mesma forma que na inserção de documentos, a remoção de documentos XML realiza, em primeira instância, a definição de uma autenticação HTTP, configurando um usuário e uma senha para a remoção do documento.

Então, é criada uma conexão HTTP, que através do método DELETE, remove o documento XML, informado na URL de conexão, da base de dados do eXist.

Os códigos fonte da aplicação e arquivo para instalação do NXD eXist estão disponíveis em http://www.inf.ufsm.br/~mmarinho/DBLPeXist.

No próximo capítulo é apresentada a conclusão do trabalho, comentando sobre os estudos feitos sobre os NXDs, a aplicação desenvolvida e possíveis melhorias para a aplicação desenvolvida.

5 CONCLUSÃO

Devido ao constante crescimento na utilização de documentos XML tanto para armazenamento quanto para troca de informações, viu-se necessário a utilização de NXDs. Então, um dos objetivos deste trabalho foi analisar cinco NXDs disponíveis para utilização gratuita e paga – eXist-db Open Source Native XML Database, Tamino XML Server, Apache Xindice, Sedna XML DBMS e SQL/XML-IMDB. Através desta análise foi possível obter uma ajuda para a necessidade de trabalhar com uma base de dados XML.

O estudo desenvolvido sobre estes NXDs teve o objetivo de obter conhecimentos sobre como cada um deles funciona, de que forma os dados são armazenados, qual a arquitetura que cada um deles utiliza, quais os requisitos para o funcionamento, se eles são proprietários ou de código aberto, quais as linguagens utilizadas para consultar os dados armazenados e também para conhecer e difundir tecnologias de armazenamento em massa que não são utilizadas com frequência, com o intuito de criar novas alternativas de escolha na hora de escolher como os dados de um sistema poderão ser armazenados.

Após os estudos feitos sobre os NXDs, foi feita a escolha de um deles, levando em consideração as informações obtidas na análise individual dos mesmos. Pelas razões discutidas no Capítulo 3, o NXD escolhido foi o eXist-db. Então foi proposto o desenvolvimento de uma aplicação para a manipulação dos dados armazenados nele.

A aplicação proposta foi desenvolvida na linguagem de programação Java e o principal objetivo dela é gerenciar o conteúdo do NXD. Com a utilização desta aplicação, o usuário pode armazenar documentos XML na base de dados do eXist, editá-los, removê-los e efetuar consultas sobre conteúdo desses documentos. Nos exemplos e nos testes da aplicação, foram utilizados alguns documentos XML de um repositório bibliográfico da Ciência da Computação chamado DBLP, porém qualquer documento XML pode ser armazenado na base de dados do eXist. Vale ressaltar que alguns documentos XML do DBLP foram modificados

para que outros testes pudessem ser efetuados, como por exemplo efetuar consultas em documentos mistos. Também foi criado um documento XML chamado *authors.xml* (figura 4.7). O objetivo deste documento é armazenar os autores presentes nos documentos XML utilizados para testes. Através do documento *authors.xml* foi possível criar uma referência para os autores no documento *dblp.xml* (figura 4.8). No documento *dblp.xml* existe alguns elementos chamados <*authorref/*>, e nesses elementos está contido o atributo *idref*, que é uma referência para o identificador do autor, atributo *id* do elemento <*author*> no documento *authors.xml*.

Sugere-se para trabalhos futuros, que seja desenvolvido uma tela onde possam ser listadas as coleções, subcoleções e os documentos contidos no NXD. Também é possível que os dados retornados das consultas, ao invés de serem apresentados na formatação XML, sejam apresentados em forma de tabela, tornando mais fácil a visualização dos dados ao usuário.

Outra melhoria a ser realizada é a inserção de DTDs as coleções do eXist, deste modo, o usuário não precisaria se preocupar em utilizar outra ferramenta para a validação do conteúdo dos documentos XML, além de manter um padrão para os documentos inseridos em coleções específicas. Também pode ser implementado um controle de inserções de documentos XML para evitar ou alertar ao usuário que documentos com mesmo nome e mesma coleção serão sobrescritos.

REFERÊNCIAS

GRAVES, M. **Designing XML Databases**. Prentice Hall PTR, 2002. Chapter 1.

W3C. **Extensible Markup Language (XML)**. Disponível em http://www.w3.org/XML. Acesso em abril de 2012.

W3C. **XML Schema**. Disponível em http://www.w3.org/XML/Schema. Acesso em abril de 2012.

W3Schools. **XML DOM Tutorial**. Disponível em http://www.w3schools.com/dom/default.asp. Acesso em abril de 2012.

Wikipedia. **Definição de Tipo de Documento**. Disponível em http://pt.wikipedia.org/wiki/Definição_de_Tipo_de_Documento. Acesso em abril de 2012.

SOUSA, F. R. C. **Replix: Um mecanismo para a replicação de dados XML**. Disponível em mdcc.ufc.br/teses/doc_download/89-089-flavio-rubens-de-carvalho-sousa. Acesso em abril de 2012.

eTutoriais. **System Architecture Overview**. Disponível em http://etutorials.org/XML/xml+data+management/Part+II+Native+XML+Databases/Chapter+3.+eXist+Native+XML+Database/3.3+System+Architecture+Overview/. Acesso em abril de 2012.

PASQUALI & DUARTE. **Estudo Comparativo dos BDXML eXist e Xindice**. Disponível em http://www.lbd.dcc.ufmg.br/colecoes/erbd/2009/006.pdf. Acesso em abril de 2012.

eXist. **eXist-db Open Source Native XML Database**. Disponível em http://exist-db.org/exist/index.xml. Acesso em abril de 2012.

eTutorials. **Tamino Architecture and APIs**. Disponível em http://etutorials.org/XML/xml+data+management/Part+II+Native+XML+Databases/Chapter+2.+Tamino-Software+AG+s+Native+XML+Server/2.2+Tamino+Architecture+and+APIs/. Acesso em abril de 2012.

SoftwareAG. **First steps with Tamino**. Disponível em http://communities.softwareag.com/ecosystem/communities/public/Developer/webmethods/products/tamino/tutorials/Tamino/index.html. Acesso de abril de 2012.

Blanken, Henk M. **Intelligent Search on XML Data**. Disponível em http://books.google.com.br/books?id=oqAu1c9MC5YC&printsec=frontcover&dq=Intelligent+Search+on+Xml+Data&hl=pt-BR&sa=X&ei=vXigT5eBGIWe8gTmzsnJAQ&ved=0CD0Q6AEwAA#v=onepage&q=Intelligent%20Search%20on%20Xml%20Data&f=false. Acesso em abril de 2012.

Tamino. **General Architecture**. Disponível em http://documentation.softwareag.com/webmethods/tamino/ins441/concepts/cfgenarc.htm. Acesso de abril de 2012.

Tamino. **Tamino XML Server – User Guide**. Disponível em http://www.comp.polyu.edu.hk/~comp532/taminoTutorial/TaminoTutorial.pdf. Acesso em abril de 2012.

MARQUES, L. M. O. **Base de Dados em Memória**. Disponível em http://www.dei.isep.ipp.pt/~paf/proj/Set2002/InMemoryDataBase.pdf. Acesso em abril de 2012.

CUONG, Nguyen V. **XML Native Database Systems – Review of Sedna, Ozone, NeoCoreXMS**. Disponível em http://swing.felk.cvut.cz/index.php?option=com_docman&task=doc_view&gid=5&Itemid=62. Acesso em abril de 2012.

Celepar. **Banco de dados, uma retrospectiva**. Disponível em http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=1364. Acesso em abril de 2012.

QuiLogic. **In Memory SQL / XML Database Technology for Universal Data Management**. Disponível em http://www.quilogic.cc/features.htm. Acesso em abril de 2012.
BOURRET, R. **XML Database Products: Native XML Databases**. Disponível em http://www.rpbourret.com/xml/ProdsNative.htm#quilogic. Acesso em abril de 2012.

Xindice. **Xindice**. Disponível em http://xml.apache.org/xindice/. Acesso em abril de 2012.

MENDES, G. **Utilizando uma Base de Dados XML Nativa aplicada ao tratamento de erros num Sistema de logs**. Disponível em http://homepages.di.fc.ul.pt/~paa/projects/conferences/coopmedia2003/09.pdf. Acesso em abril de 2012.

FIGUEIREDO, F. J. V. **XML E BANCOS DE DADOS**. Disponível em http://meusite.mackenzie.com.br/rogerio/tgi/2003XMLXindex.PDF. Acesso em abril de 2012.

OLIVEIRA, A. M. C. **SAVE: SISTEMA DE ATENDIMENTO AO VESTIBULANDO UTILIZANDO BANCO DE DADOS NATIVO XML**. Disponível em http://projetos.inf.ufsc.br/arquivos_projetos/projeto_349/Monografia Alex.pdf. Acesso em abril de 2012.

W3Schools. **XPath Tutorial**. Disponível em http://www.w3schools.com/xpath/. Acesso em maio de 2012.

W3Schools. **XQuery Tutorial**. Disponível em http://www.w3schools.com/xquery/. Acesso em maio de 2012.

X-Query. **XML Query Language: X-Query**. Disponível em http://www.comp.polyu.edu.hk/~comp532/taminoTutorial/pdf/xql.pdf. Acesso em maio de 2012.

X-Query. **Query Language? Tamino X-Query**. Disponível em http://etutorials.org/XML/xml+data+management/Part+II+Native+XML+Databases/Chapter+2.+Tamino-Software+AG+s+Native+XML+Server/2.4+Querying+XML/. Acesso em maio de 2012.

IBM. **Managing XML data: eXist -- an open source native XML database**. Disponível em http://www.ibm.com/developerworks/library/x-mxd5/#ibm-pcon. Acesso em maio de 2012.

DBLP. **Digital Bibliography & Library Project**. Disponível em http://www.informatik.uni-trier.de/~ley/db/. Acesso em maio de 2012.

NETBEANS. Disponível em: http://netbeans.org. Acesso em maio de 2012.