JULIANA VERMELHO MARTINS

INFERÊNCIA DE MODELOS DE DADOS E CRIAÇÃO DE BASES DE DADOS ESTRUTURADAS A PARTIR DE DOCUMENTOS SEMI-ESTRUTURADOS

Dissertação apresentada à Pontifícia Universidade Católica do Paraná para a obtenção do título de Mestre em Informática Aplicada.

Área de Concentração:
Sistemas de Informação

Orientador: Prof. Dr. Celso Antonio Alves Kaestner

CURITIBA

2001

Martins, Juliana Vermelho
Inferência de Modelos de Dados e Criação de Bases de Dados Estruturadas a Partir de Documentos Semi-Estruturados. Curitiba, 2001.
88p.

Dissertação – Pontifícia Universidade Católica do Paraná. Programa de Pós-Graduação em Informática Aplicada.

1. Dados semi-estruturados 2. Banco de dados 3. Linguagem HTML I.Pontifícia Universidade Católica do Paraná. Centro de Ciências Exatas e de Tecnologia. Programa de Pós-Graduação em Informática Aplicada II-t

Pontifícia Universidade Católica do Paraná

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Informática Aplicada

ATA DA SESSÃO PÚBLICA DE DEFESA DE DISSERTAÇÃO DE MESTRADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA APLICADA DA PONTIFÍCIA UNIVERSIDADE CATÓLICA DO PARANÁ

DEFESA DE DISSERTAÇÃO Nº 37

Aos 21 dias do mês de setembro de 2001 realizou-se a sessão pública de defesa da dissertação "**Inferência de Modelos de Dados e Criação de Bases de Dados Estruturadas a Partir de Documentos Semi-Estruturados**", apresentada por **Juliana Vermelho Martins** como requisito parcial para a obtenção do título de **Mestre em Ciências**, perante uma Banca Examinadora composta pelos seguintes membros:

	assinatura	parecer
Prof. Dr. Celso Kaestner PUCPR (Presidente)		APROVADO
Prof. Dr. Alex A. de Freitas PUCPR	Alex Alves Freitas	APROVADO
Prof. Dr. Marcos A. H. Shmeil PUCPR		aprovado
Prof. Dr. Carlos A. Heuser UFRGS		aprovado

Conforme as normas regimentais do PPGIA e da PUCPR, o trabalho apresentado foi considerado aprovado sem restrições (aprovado sem restrições, aprovado com exigências ou reprovado), segundo a avaliação da maioria dos membros da Banca Examinadora acima indicada.

Observações, exigências e/ou restrições da Banca Examinadora, quando houver:

_____

_____

continuar no verso, se necessário

Prof. Dr. Carlos Alberto Maziero, Diretor
Programa de Pós-Graduação em Informática Aplicada
Pontifícia Universidade Católica do Paraná

Para Vidal e Felipe,

meus amores.

AGRADECIMENTOS

Ao Professor Celso Kaestner por toda ajuda, apoio e infinita paciência durante a execução deste trabalho.

Ao Professor Cláudio de Oliveira pela importante contribuição na idéia de estruturação da implementação.

A Vidal Martins por todo carinho, compreensão, companheirismo e estímulo em todo o tempo de duração do curso.

Ao meu querido filho Felipe que suportou todas as ausências sem ainda poder compreendê-las.

Às várias pessoas que cuidaram do Felipe para que eu pudesse concluir este trabalho.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

DER	DIAGRAMA ENTIDADE RELACIONAMENTO
DDL	DATA DEFINITION LANGUAGE
DML	DATA MANIPULATION LANGUAGE
HTML	HYPERTEXT MARKUP LANGUAGE
OEM	OBJECT EXCHANGE MODEL
SGBD	SISTEMA GERECIADOR DE BANCO DE DADOS
SGML	STANDARD GENERALIZED MARKUP LANGUAGE
SQL	STRUCTURED QUERY LANGUAGE
UML	UNIFIED MODELING LANGUAGE
WWW	WORLD WIDE *WEB*
XML	EXTENSIBLE MARKUP LANGUAGE

RESUMO

A World Wide Web tem se configurado cada vez mais como grande repositório de informações. Diversas pesquisas têm sido realizadas no sentido de consultar essas informações diretamente da *web*. Entretanto, como o conteúdo das páginas da Internet é atualizado constantemente, perde-se a evolução histórica desses dados, já que tais consultas são realizadas sobre a configuração atual das páginas. Além disso, os documentos existentes na *web* não possuem um esquema rígido que os descreva, logo não estão armazenados de acordo com as regras dos bancos de dados relacionais, o que dificulta a utilização de mecanismos como a linguagem SQL.

O objetivo desse trabalho é desenvolver um método para inferir a estrutura de um documento HTML e armazenar o conteúdo desse documento em uma base de dados relacional temporal a qual será preenchida com sucessivas leituras posteriores do documento. Tal base poderá depois ser consultada por meio de mecanismos convencionais, como, por exemplo, linguagem SQL ou utilizada em como entrada para posterior aproveitamento em aplicações de *data mining*.

Palavras-Chaves: 1. Dados semi-estruturados 2. Banco de dados 3. Linguagem HTML

RESUMO

A World Wide Web tem se configurado cada vez mais como grande repositório de informações. Diversas pesquisas têm sido realizadas no sentido de consultar essas informações diretamente da *web*. Entretanto, como o conteúdo das páginas da Internet é atualizado constantemente, perde-se a evolução histórica desses dados, já que tais consultas são realizadas sobre a configuração atual das páginas. Além disso, os documentos existentes na *web* não possuem um esquema rígido que os descreva, logo não estão armazenados de acordo com as regras dos bancos de dados relacionais, o que dificulta a utilização de mecanismos como a linguagem SQL.

O objetivo desse trabalho é desenvolver um método para inferir a estrutura de um documento HTML e armazenar o conteúdo desse documento em uma base de dados relacional temporal a qual será preenchida com sucessivas leituras posteriores do documento. Tal base poderá depois ser consultada por meio de mecanismos convencionais, como, por exemplo, linguagem SQL ou utilizada em como entrada para posterior aproveitamento em aplicações de *data mining*.

Palavras-Chaves: 1. Dados semi-estruturados 2. Banco de dados 3. Linguagem HTML

ABSTRACT

The World Wide Web represents today a large information repository. Several project have proposed some ways of querying this information directly from the web. However, as the contents of the Internet is constantly updated the historical evolution of these data is lost, since queries are performed over the page's current configuration. Besides, web documents do not have a rigid schema describing them, therefore they are not stored according to the relational data model rules. This makes difficult the use of mechanisms like the SQL language.

The aim of this work is to develop a method to infer the structure of na HTML document and to store the contents of such document in a temporal relational database. Other versions of this document will be used in order to fill the database. These data could be queried by conventional mechanisms like, for example, the SQL language or used as input in future data mining applications.

Keywords: 1. Semistructured data. 2. Databases. 3. HTML language.

ABSTRACT

The World Wide Web represents today a large information repository. Several project have proposed some ways of querying this information directly from the web. However, as the contents of the Internet is constantly updated the historical evolution of these data is lost, since queries are performed over the page's current configuration. Besides, web documents do not have a rigid schema describing them, therefore they are not stored according to the relational data model rules. This makes difficult the use of mechanisms like the SQL language.

The aim of this work is to develop a method to infer the structure of na HTML document and to store the contents of such document in a temporal relational database. Other versions of this document will be used in order to fill the database. These data could be queried by conventional mechanisms like, for example, the SQL language or used as input in future data mining applications.

Keywords: 1. Semistructured data. 2. Databases. 3. HTML language.

1 INTRODUÇÃO

A *World Wide Web* tornou-se um importante veículo para disseminar informação. O número de grandes sites *web* com estrutura complexa e cuja informação deriva de fontes múltiplas de dados está aumentando [13]. Como conseqüência, o surgimento de novas tecnologias para suporte à *web* tem expandido significativamente os tipos de dados disponíveis em sistemas de gerenciamento de informação.

É sabido que os sistemas gerenciadores de bancos de dados (SGBDs) fornecem tecnologia para consultas e manutenção de dados altamente estruturados de maneira flexível e eficiente, mas a maioria das informações na *web* apresenta-se na forma de textos, arquivos de áudio e figuras, os quais não são considerados dados estruturados por não terem um esquema que os descreva. Recentemente, a manipulação desse tipo de informação menos estruturada também tornou-se campo de grande interesse.

Em um banco de dados semi-estruturado não existe um esquema fixo para a sua estrutura, o que faz com que a manipulação dos seus dados seja não trivial. O desejo de tratar a *web* como um banco de dados gerou várias pesquisas no sentido de utilizar técnicas de banco de dados em dados semi-estruturados [16].

Estender técnicas de bancos de dados a documentos *web* apresenta uma série de novos desafios. Primeiro, são necessárias ferramentas para explorar os imensos grafos de páginas e localizar as informações de interesse. Então, uma vez que documentos que contenham tais informações tenham sido encontrados, um problema chave consiste em identificar partes relevantes dos dados dentro do texto e extraí-las com o objetivo de construir uma representação de banco de dados em algum modelo de dados, o qual possa, então, ser manipulado por uma linguagem de consulta. Este processo

é freqüentemente referenciado como fazer o *wrapping* de uma fonte de dados [22].

Ao analisar todos esses aspectos, julgou-se interessante elaborar um projeto de pesquisa que viesse a contribuir com outras pesquisas já em andamento, no sentido de buscar informações na *web*, inferir a estrutura das páginas e converter seu conteúdo para uma base de dados que possa ser posteriormente consultada e utilizada pelos usuários em um sistema de banco de dados.

Os esforços foram concentrados não na busca de documentos na *web*, já tema de diversas pesquisas [20][14][19][25][9][7], mas sim no reconhecimento da estrutura de um documento em linguagem HTML e na respectiva extração dos dados contidos nesse documento e respectiva conversão dos mesmos em uma base de dados relacional. O objetivo deste trabalho é mostrar que é possível essa conversão, e que a base de dados gerada pode depois ser utilizada para consultas com linguagens de banco de dados, mais especificamente, linguagem SQL. A base também poderá ser utilizada como parte de um sistema de *data mining* no futuro, mas essa aplicação está fora do escopo do presente trabalho.

Aplicações em potencial para esta tecnologia poderiam ser aquelas relacionadas a observações meteorológicas e também ao mercado financeiro. Em ambos os casos, seria possível obter uma base de dados histórica tanto das condições climáticas de uma dada região quanto dos indicadores financeiros de um determinado mercado. A vantagem reside na possibilidade de análises de tendências feitas sobre dados reais, obtidos diretamente de uma fonte confiável (*site* meteorológico ou *site* de bolsa de valores). Também é relevante o fato de que essa tecnologia pode ser utilizada como monitoramento de sites cujo conteúdo não é diretamente disponibilizado na forma de base de dados. Dessa forma, um usuário interessado em *espionar* a

variação de preços de um concorrente, por exemplo, poderia utilizar a técnica aqui apresentada para obter uma base de dados com a política de preços adotada por esse concorrente durante um período de tempo.

1.1 SISTEMAS QUE TRATAM DADOS SEMI-ESTRUTURADOS

Para que esse trabalho pudesse ser realizado, primeiramente foi feita uma pesquisa de projetos semelhantes que estavam sendo feitos em grandes universidades.

O ponto de partida para esse estudo foi o trabalho desenvolvido pela Universidade Federal do Rio Grande do Sul, sob a coordenação do professor Carlos Alberto Heuser [16].

Esses projetos assumem o fato de que páginas *web* são documentos não estruturados ou semi-estruturados. Um documento semi-estruturado é aquele que não possui um esquema que o descreva, indicando que não há separação entre a descrição da estrutura e o próprio dado [2]. Verificou-se que os estudos em dados semi-estruturados atuam em uma ou mais das seguintes direções: modelagem de dados semi-estruturados, linguagem de consulta a dados semi-estruturados e inferência de estrutura em dados semi-estruturados, descritos a seguir.

O modelo para dados semi-estruturados mais aceito e utilizado é aquele desenvolvido pela Universidade de Stanford dentro do projeto TSIMMIS, conhecido como OEM (*Object Exchange Model*) [26]. Os dados representados nesse modelo são auto-descritos, no sentido de que podem ser compreendidos sem que seja necessária nenhuma referência a qualquer esquema externo. No modelo OEM existem objetos, os quais possuem quatro componentes: .

(i) **Object ID**: expressão que descreve de onde vem o objeto.

(ii) **Label**: indica o que o objeto representa. Pode ser entendido

como a qual *classe* o objeto pertence.

(iii) **Type**: o tipo de seu valor. Pode ser do tipo atômico, por exemplo, *string*, ou do tipo *conjunto*.

(iv) **Value**: o conteúdo do objeto, o qual pode ser atômico, ou outro objeto.

Outro campo de estudo em relação aos dados semi-estruturados diz respeito às linguagens de consulta a dados desse tipo. Há duas abordagens possíveis: acrescentar funcionalidades à linguagem SQL para que ela seja capaz de acessar dados semi-estruturados ou criar uma linguagem baseada nas noções de dados semi-estruturados para depois ajustá-la a questões relativas a performance e sintaxe.

Como exemplo da primeira abordagem, pode-se citar a linguagem WebSQL [24]. Essa linguagem tem por objetivo consultar a *web* por meio de comandos *SQL-like.* Já a linguagem UnQL [8] faz consultas a dados organizados em grafos dirigidos, com um dos nós representando a raiz do grafo e cujas arestas sejam rotuladas, forma bastante utilizada para representação de dados semi-estruturados.

Outros exemplos de linguagens para consulta a dados semi-estruturados são Lorel [3] que é parte integrante do projeto TSIMMIS [26][15] e faz uma extensão da linguagem OQL (linguagem de consulta para bancos de dados orientados a objetos) e StruQL, parte do projeto STRUDEL [13] que também utiliza a noção de grafos rotulados e dirigidos.

O último direcionamento de pesquisas em relação a dados semi-estruturados está ligado à inferência de uma possível estrutura que exista por trás desses dados. Dois projetos relevantes nessa área são o TSIMMIS [26][15] e o NoDoSe [4], os quais, além da inferência também se preocupam com a extração dos dados dos documentos.

O primeiro extrai informações de documentos HTML e gera objetos

segundo o modelo OEM. Entretanto, a descrição da estrutura do documento HTML deve ser feita manualmente. O segundo é uma ferramenta para determinar, de maneira semi-automática, a estrutura de documentos além de extrair seus dados. No projeto NoDoSe são levados em consideração não só documentos no formato HTML como também no formato texto.

O escopo do trabalho foi definido tomando por base estes três encaminhamentos possíveis para o estudo de dados semi-estruturados: modelagem de dados semi-estruturados, linguagem de consulta a dados semi-estruturados e inferência de estrutura em dados semi-estruturados.

Como afirmado anteriormente, vários dos projetos que se dedicam ao estudo de dados semi-estruturados utilizam o modelo OEM para descrição dos dados. Nesta dissertação essa tecnologia não foi empregada porque o principal objetivo era extrair os dados de documentos semi-estruturados e armazená-los em bancos de dados relacionais, os quais pudessem ser consultados com linguagem SQL. Isso levou ao desenvolvimento de uma abordagem particular.

Em relação às linguagens para consulta a dados semi-estruturados, estabeleceu-se que não é objetivo desse trabalho realizar consultas diretamente sobre dados semi-estruturados. Ao contrário, desejou-se converter esses dados para dados estruturados, a fim de utilizar linguagem SQL para fazer as consultas.

Quanto à inferência da estrutura das páginas, definiu-se que esta deveria ser feita preferencialmente de maneira automática, sem que o usuário tivesse a necessidade de qualquer conhecimento sobre o código HTML associado à página.

A metodologia de trabalho empregada consistiu no estudo das funções dos rótulos HTML e na elaboração de uma estratégia de transformação do conteúdo de um documento HTML numa base de dados

temporal. A fim de verificar se essa estratégia era conveniente e produziria um resultado adequado, desenvolveu-se um protótipo, o qual tem por objetivo exatamente o que propõe essa dissertação: *a inferência de modelos de dados e a criação de uma base de dados estruturada a partir de dados semi-estruturados*. O protótipo que foi desenvolvido, bem como os resultados alcançados, serão descritos neste trabalho.

Este trabalho está organizado da seguinte forma. O capítulo 2 descreve algumas das técnicas utilizadas na elaboração do projeto de pesquisa. O capítulo 3 descreve detalhadamente o protótipo que foi implementado. O capítulo 4 discute os resultados obtidos. Por fim, o capítulo 5 apresenta conclusões e algumas possibilidades de desenvolvimento futuro.

2 TÉCNICAS UTILIZADAS

Esta seção faz uma fundamentação de alguns formalismos empregados na construção do protótipo. Da mesma forma serão feitos comentários sobre as notações dos diagramas utilizados na sua descrição.

2.1 LINGUAGEM HTML

A linguagem HTML é subproduto da linguagem SGML e foi desenvolvida com o objetivo de publicar documentos na *web*. Tais documentos podem ser compostos de textos, tabelas, figuras e outros itens multimídia. Ela permite a conexão entre documentos ou partes de documentos por meio de *links* [29].

HTML é uma linguagem de marcação. Autores de documentos HTML inserem marcas no mesmo com o objetivo de representar informações estruturais e de apresentação ao longo de seu conteúdo. Tais marcas são implementadas por meio de *elementos*. Um elemento é composto por um rótulo inicial, um conteúdo e um rótulo final. Exemplo:

```
<P>Isto é um parágrafo.</P>
```

No exemplo apresentado, <P> representa o rótulo inicial, o texto *Isto é um parágrafo* representa o conteúdo e o rótulo </P>, o rótulo final. Embora essa seja uma estrutura seguida pela maioria dos elementos de marcação HTML, vários quebram essa regra. Em alguns elementos o rótulo final é opcional, em outros, o rótulo inicial é opcional. Há ainda elementos que não possuem conteúdo, como é o caso do rótulo
 que representa uma quebra de linha. Nesses caso, o rótulo final também não existirá.

Na literatura pesquisada [28], o termo rótulo é utilizado como sinônimo de elemento. Entretanto, a especificação HTML chama atenção para o fato de que esses termos não são sinônimos [29]. Neste trabalho, sempre que for utilizada a expressão *elemento* será como referência ao conjunto rótulo inicial, conteúdo e rótulo final. Quando a expressão utilizada for simplesmente rótulo, a referência será somente a um dos rótulos componentes do elemento (inicial ou final). Caso o rótulo possua o caractere / no seu início, será um rótulo final. Em caso contrário, será um rótulo inicial.

Rótulos iniciais também podem conter atributos. Por exemplo:

```
<TABLE BORDER=5 COLS=7>
```

Esse rótulo define uma tabela que possui borda com 5 *pixels* e 7 colunas em suas linhas.

Apenas um subconjunto dos rótulos HTML foi selecionado para o reconhecimento da estrutura de um documento. Quais são esses rótulos e o motivo de eles terem sido escolhidos será discutido no decorrer deste trabalho.

2.2 BANCOS DE DADOS

A proposta desse projeto é a criação de uma base de dados que contenha o conteúdo de sucessivas leituras de um documento HTML. Portanto, um modelo de banco de dados deveria ser escolhido para realizar esse armazenamento. O modelo selecionado foi o relacional.

Entretanto, um aspecto importante dos dados selecionados é o tempo em que eles foram obtidos. Portanto, aspectos relacionados à dimensão temporal dos dados foram levados em consideração.

Os dois modelos de bancos de dados, relacional e temporal, serão discutidos brevemente nas próximas seções.

2.2.1 Banco de Dados Relacional

A seção seguinte faz uma fundamentação teórica básica de bancos de dados relacionais, modelo escolhido para a implementação do protótipo e justifica essa escolha. O texto que segue está fortemente baseado em [10].[1]

Os sistemas relacionais têm sua fundamentação teórica formal no *modelo relacional de dados*. Esse modelo trata de três aspectos:

(i) **Estrutural**: os dados são percebidos pelo usuário como tabelas.

(ii) **Integridade**: as tabelas satisfazem a restrições de integridade.

(iii) **Manipulativo**: os operadores utilizados para manipular essas tabelas derivam tabelas de outras tabelas.

As tabelas formam a estrutura lógica (e não física) de um sistema relacional. No nível físico, cada fabricante de SGBD determina como serão armazenados os dados. Desde que, para o usuário, os dados sejam percebidos como tabelas, não há restrição ou imposição sobre o modo como esse armazenamento é feito.

Os aspectos abordados anteriormente permitem a afirmação que diz que bancos de dados relacionais devem obedecer a um princípio, chamado **Princípio da Informação**: "*todo o conteúdo de informação do banco de dados é representado de um e somente um modo, ou seja, como valores explícitos em posições de colunas em linhas de dados*"[10].

A especificação *relacional* associada a essa tecnologia reside no fato de que *relação* é o termo matemático equivalente a *tabela*. Informalmente, os termos são considerados sinônimos e, mais comumente, é o termo *tabela* que aparece nos textos teóricos e no jargão empregado pelos profissionais de informática. Neste trabalho, será utilizado o termo *relação* sempre que se for referenciada uma tabela de banco de dados. A expressão *tabela* ficará restrita

[1] Essa fundamentação teórica não pretende, de maneira nenhuma, esgotar o assunto que é bastante vasto. Ela se prende apenas nos aspectos relevantes para a implementação do protótipo.

à identificação de um elemento passível de ser encontrado em um código HTML.

Em relação ao aspecto **estrutural** de um banco de dados relacional, há alguns termos que devem ser especificados. A figura a seguir mostra tais termos:

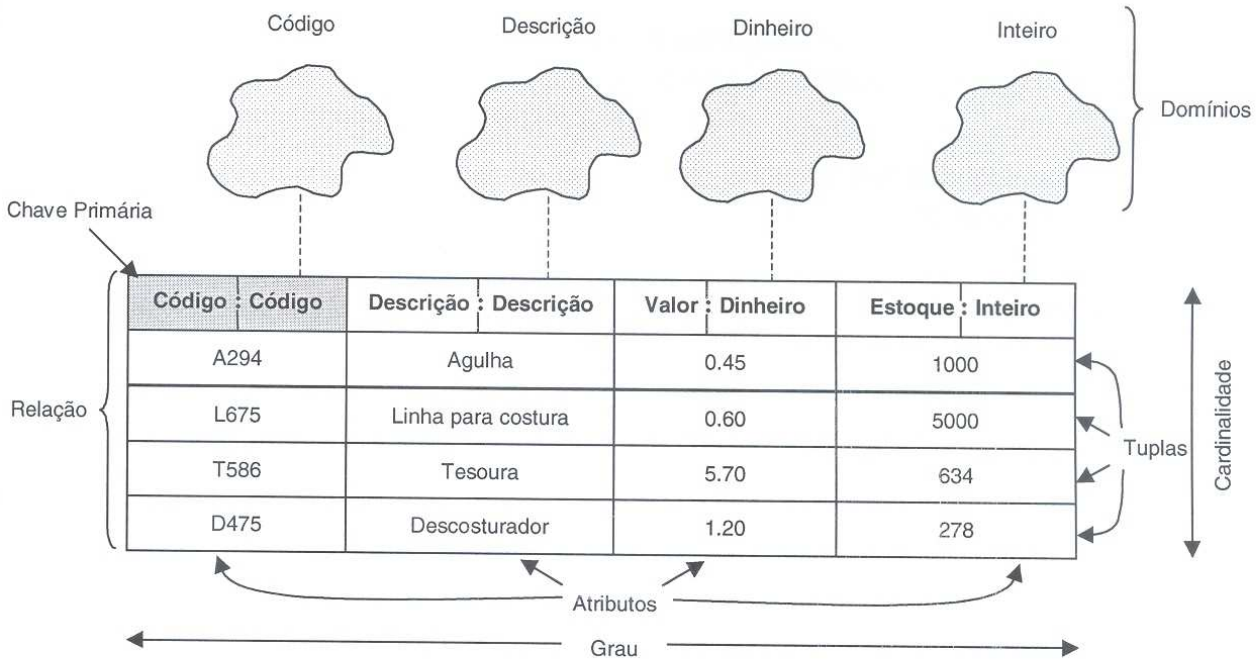


FIGURA 1 – TERMINOLOGIA ESTRUTURAL PARA O MODELO RELACIONAL

Assumindo que os termos relação e tabela sejam sinônimos, pode-se depreender que *tupla* é equivalente a uma linha de uma tabela e *atributo* corresponde a uma coluna dessa mesma tabela. A quantidade de tuplas existente representa a *cardinalidade* e a de atributos, o *grau*. Um domínio é um conjunto de valores que agrupa aqueles possíveis para um determinado atributo. Por exemplo, dentre todos os valores possíveis para código, o material cuja descrição é *Agulha,* possui o valor *A294*. A própria descrição do material foi retirada de um domínio, assim como todos os outros atributos. Um domínio é, em última análise, um tipo de dados, o qual pode ser definido pelo sistema ou pelo usuário.

Feita essa apresentação inicial dos conceitos relacionados ao aspecto estrutural de um modelo relacional, é possível uma definição formal de **relação**.

> "Dada uma coleção de *n* tipos ou domínios *Ti* (*i* = 1, 2, ..., *n*), não necessariamente todos distintos, *r* será a **relação** sobre esses tipos se consistir em duas partes, um *cabeçalho* e um *corpo*, onde:
>
> - O **cabeçalho** é um conjunto de *n* **atributos** da forma *Ai:Ti*, onde *Ai* (que devem ser todos distintos) são os *nomes de atributos* de *r* e *Ti* são os *nomes de tipos* correspondentes (*i* = 1, 2, ..., *n*).
> - O **corpo** é um conjunto de *m* **tuplas** *t*, onde *t* é por sua vez um conjunto de componentes da forma *Ai:vi* no qual *vi* é um valor do tipo *Ti* – o *valor de atributo* correspondente ao atributo *Ai* da tupla *t* (*i* = 1, 2, ..., *n*).
>
> Os valores *m* e *n* são chamados respectivamente **cardinalidade** e **grau** da relação *r*." [10]

As relações apresentam algumas propriedades decorrentes de sua definição. São elas:

(i) *Não existem tuplas em duplicata*. A duplicidade de tuplas não ocorre porque o corpo de uma relação é um conjunto matemático e conjuntos desse tipo não incluem repetições.

(ii) *Tuplas não são ordenadas de cima para baixo*. Tal propriedade se explica pelo mesmo motivo que a anterior: o corpo de uma relação é um conjunto matemático e conjuntos desse tipo não consideram a ordem entre seus elementos.

(iii) *Atributos não são ordenados da esquerda para a direita*. Essa propriedade advém do fato de que o cabeçalho de uma relação também é um conjunto matemático, o qual, como já mencionado, não considera a ordem entre seus elementos.

(iv) *Cada tupla contém exatamente um valor para cada atributo*. A definição de uma tupla diz que ela é um conjunto de componentes da forma *Ai:vi* (*i* = *1, 2, ..., n*), portanto, cada atributo possui um único valor. "Uma relação que satisfaz a essa quarta propriedade

é dita normalizada ou, de forma equivalente, diz-se que ela está na **primeira forma normal**". [10]

Para finalizar essa discussão teórica sobre bancos de dados relacionais, serão apresentados os conceitos de **chaves**.

A primeira propriedade apresentada para o modelo relacional estabeleceu que não pode haver tuplas em duplicata em uma relação, o que significa dizer que tuplas têm a propriedade de **unicidade**. Pode ocorrer, entretanto, que um subconjunto dos atributos de uma tupla tenha, ele também, essa propriedade de unicidade. Esse fato faz parte da definição de *chave candidata*:

> "Seja *K* um conjunto de atributos da variável de relação *R*. Então, *K* é uma **chave candidata** para *R* se e somente se ela possui ambas as propriedades a seguir:
>
> a) **Unicidade**: nenhum valor válido de *R* contém duas tuplas diferentes com o mesmo valor para *K*.
>
> b) **Irredutibilidade**: nenhum subconjunto próprio de *K* tem a propriedade de unicidade." [10]

As chaves candidatas fornecem o mecanismo pelo qual tuplas de uma relação podem ser endereçadas e são fundamentais para que um sistema relacional funcione de acordo com suas definições.

Relações podem ter mais de uma chave candidata. Nesses casos, o modelo relacional tem exigido a escolha de uma dessas chaves candidatas a qual será a **chave primária** da relação. As outras serão suas **chaves alternativas**.

Além das chaves mencionadas anteriormente, é necessário conceituar **chave estrangeira**. De maneira simplificada, uma chave estrangeira é um conjunto de atributos de uma relação *R2* cujos valores devem corresponder a algum valor existente em alguma chave candidata de uma relação *R1*. Formalmente, uma chave estrangeira é definida da maneira que segue:

"Seja *R2* uma variável de relação. Então, uma **chave estrangeira** em *R2* é um conjunto de atributos de *R2*, digamos *FK*, tal que:

a) Existe uma variável de relação *R1* (*R1* e *R2* não necessariamente distintas) com uma chave candidata CK, e

b) Para sempre, cada valor de *FK* no valor atual de *R2* é idêntico ao valor de *CK* em alguma tupla no valor atual de *R1*." [10]

Uma série de conseqüências decorre das definições de chave candidata, primária e estrangeira, mas elas ou são bastante conhecidas pela comunidade acadêmica, ou são irrelevantes para o escopo deste trabalho. Uma discussão aprofundada dessas conseqüências pode ser obtida em [10].

A tecnologia relacional foi escolhida pelo fato de estar bastante madura e implementada em produtos comerciais de fácil acesso. Entretanto, nem todos os aspectos do modelo relacional da forma como estão apresentados na versão mais recente descrita em [10] estão presentes nos produtos comerciais, mais especificamente, o tratamento dado ao domínio de um atributo. No momento oportuno essa diferença entre a teoria proposta e a efetiva implementação será discutida, bem como a solução adotada.

2.2.2 Bancos de dados temporais

Uma necessidade das aplicações que não pode ser completamente atendida pela tecnologia relacional é a manipulação de informações históricas dos dados armazenados em um banco de dados. Por esse motivo, pesquisas foram desenvolvidas no sentido de definir conceitos e estratégias para tratar esse tipo de informação. Tais pesquisas determinaram o surgimento da tecnologia de Bancos de Dados Temporais [12].

Alguns conceitos desses estudos, relevantes para a realização desse trabalho, serão abordados a seguir:

- **Dimensão temporal**: os bancos de dados relacionais possuem somente duas dimensões: as instâncias dos dados (linhas da

tabela) e os atributos de cada instância (coluna da tabela). Os modelos temporais acrescentam mais uma dimensão aos modelos tradicionais – a dimensão temporal. Essa nova dimensão associa alguma informação temporal a cada valor. Caso o valor de um atributo seja alterado, o valor anterior não é removido do banco de dados – o novo valor é acrescentado, associado a alguma informação que define, por exemplo, seu tempo inicial de validade. Todos os valores definidos ficam armazenados no banco de dados.

- **Tempo absoluto e tempo relativo**: tempo absoluto declara um instante específico, definido com uma granularidade determinada, associado a um fato. Exemplo: a data de nascimento de uma pessoa. O tempo relativo relaciona sua validade à validade de um outro fato, ou ao momento atual. Exemplo: o salário aumentou ontem, a loja abriu dois meses depois da inauguração do shopping.
- **Variação temporal**: o tempo pode ser considerado contínuo ou discreto, embora suponha-se que o tempo seja contínuo por natureza. Entretanto, representando o tempo de forma discreta, pode-se simplificar consideravelmente a implementação de modelos de dados. Quando considerada a variação temporal discreta, a definição de informações sob o ponto de vista da sua validade pode ser feita de três formas: *variação ponto a ponto* (o valor definido vale somente no ponto em que foi determinado), *variação por escada* (o valor definido vale até que outro valor seja determinado) e *variação definida por uma função* (uma função define os valores e permite a interpolação para obter os valores nos pontos não determinados).

Alguns elementos primitivos são necessários para a representação temporal. Entre todos os que são considerados nessa tecnologia serão destacados os seguintes:

- **Granularidade temporal**: consiste na duração de um *chronon* (intervalo temporal que não pode ser decomposto). Mais de uma granularidade pode ser considerada simultaneamente em um mesmo sistema (minutos, dias, anos, etc.).
- **Instante no tempo**: representa um ponto no tempo e depende da forma de variação temporal considerada. Se a variação considerada é contínua o instante é um ponto no tempo de duração infinitesimal. Por outro lado, se for discreta, um instante é representado por um dos *chronons* da linha do tempo suportada pelo modelo. Há também a existência de um instante especial correspondente ao **instante atual**, o qual se move constantemente ao longo do eixo do tempo.
- **Representação temporal explícita e implícita**: a representação é explícita quando é associado um valor temporal a uma informação na forma de um rótulo temporal (*timestamping*) e é implícita quando é utilizada uma linguagem de lógica temporal.
- **Tempo de transação e tempo válido:** o tempo de transação é aquele no qual o fato é registrado no banco de dados. Tempo de validade é aquele em que o valor é válido na realidade modelada. Há ainda o **tempo definido pelo usuário** que consiste de propriedades temporais definidas explicitamente e manipuladas pelos programas de aplicação.

"Um banco de dados temporal é aquele que apresenta alguma forma implícita de representação de informações temporais em que podem ser utilizados o tempo de transação e/ou de validade para representar esta

informação temporal." [12]. Com base nisso pode-se classificar os bancos de dados de quatro formas, as quais serão apresentadas a seguir.

- **Bancos de dados instantâneos**: correspondem aos bancos de dados convencionais, em que somente os valores presentes são armazenados. A cada modificação de valor de um atributo o valor antigo é perdido e somente o último fica disponível. Caso se deseje fazer qualquer manutenção de informações temporais essas deverão ser feitas explicitamente através de atributos inseridos nas respectivas tabelas.
- **Bancos de dados de tempo de transação**: nesse tipo de banco de dados, a cada atualização do mesmo será associado automaticamente pelo SGBD um rótulo de tempo, conhecido como tempo de transação. Esse valor é atribuído pelo sistema e o usuário não tem qualquer gerência sobre ele. Nesse tipo de banco de dados, o estado atual é composto pelas últimas alterações que foram feitas nos valores. Caso se deseje qualquer informação de tempo de validade, esta deverá ser incluída explicitamente como atributo nas tabelas.
- **Bancos de dados de tempo de validade**: o tempo de validade é definido pelo usuário e pode representar o início de sua validade, a validade somente em determinado ponto no tempo ou seu intervalo de validade. Neste tipo de banco de dados, não se tem acesso ao tempo em que as informações foram definidas, ou seja, não há acesso ao tempo de transação, mas aqui pode-se corrigir informações do passado, caso elas tenham sido registradas incorretamente.
- **Bancos de dados bitemporais**: é a forma mais completa de armazenar os dados temporais, já que tempo de transação e

também de validade são associados a cada dado. Com isso, toda a história do banco de dados fica armazenada. O estado atual é composto pelos valores atualmente válidos e o estado futuro pode ser definido por meio do tempo de validade.

Há três formas de representar temporização em modelos relacionais. Pode-se associar o tempo:

(i) ao banco de dados como um todo, quando cada estado do banco de dados é armazenado por completo, com rótulo temporal. Cada alteração elementar do banco de dados cria um novo estado,

(ii) às relações, quando cada relação é temporizada e para cada estado dessa relação todas as tuplas devem ser armazenadas com o rótulo temporal, e

(iii) às tuplas, quando cada tupla é temporizada e uma alteração elementar de valores de uma relação define uma nova tupla e somente essa precisa ser armazenada.

Para a implementação temporal do protótipo desenvolvido nesta dissertação, foram escolhidos os seguintes critérios: representação temporal explícita com variação temporal discreta ponto a ponto. O banco de dados será do tipo tempo de transação com o caráter temporal associado às tuplas. Tal representação temporal foi escolhida por atender plenamente às características relacionadas ao tempo necessárias para identificar as várias versões de um mesmo documento HTML.

2.3 LINGUAGEM SQL

A linguagem SQL foi desenvolvida a partir de 1974 como complemento à tecnologia relacional de banco de dados e com o objetivo de realizar consultas em dados armazenados em bancos de dados desse tipo [17]. Atualmente, essa linguagem é padronizada por um comitê da ISO [18].

Embora seja definida como linguagem de consulta, ela possui capacidades que vão além disso, possibilitando a definição de esquemas de bancos de dados, a manipulação dos dados em todos os níveis (inclusão, alteração e consulta), além de definições de segurança dos dados.

Nesta dissertação serão utilizados os seguintes comandos que fazem parte da DDL e da DML, respectivamente, Linguagem de Definição de Dados e Linguagem de Manipulação de Dados:

- CREATE TABLE (DDL): comando que cria uma tabela em um banco de dados.
- INSERT INTO (DML): comando que insere uma tupla ou um conjunto de tuplas em uma tabela de banco de dados.
- SELECT (DML): comando que recupera informações de uma ou mais tabelas de um banco de dados.

2.4 LINGUAGEM UML

A tecnologia utilizada para representar a estrutura reconhecida a partir de um documento HTML foi a orientação a objetos. Para representar objetos, classes, herança, relações entre objetos e outros aspectos dessa tecnologia foi utilizada a linguagem UML [6]. Essa linguagem foi escolhida por ser aquela adotada pelo grupo OMG (*Object Management Group*) [27], responsável pela padronização na área de objetos.

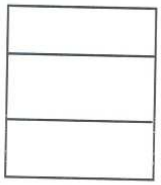
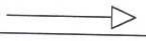
De todos os modelos propostos por essa linguagem, três serão utilizados na continuação desse documento. A fim de que eles possam ser corretamente interpretados, serão fornecidas descrições breves do objetivo de cada um dos diagramas, bem como dos símbolos empregados.

2.4.1 Diagrama de classes

Mostra um conjunto de classes e os relacionamentos existentes entre elas, além de, opcionalmente, os atributos e métodos de cada classe.

Os símbolos utilizados nesse modelo estão descritos na tabela a seguir:

TABELA 1 – SÍMBOLOS UTILIZADOS EM UM DIAGRAMA DE CLASSES

Símbolo	Significado
	Representa uma classe. Na parte superior do compartimento aparece o nome da classe. Na parte intermediária, seus atributos e na parte inferior, seus métodos.
	Representa uma agregação entre duas classes. A classe que está do lado em que existe o losango *contém* a classe que está do outro lado da associação. O losango preenchido significa uma associação por referência, ou seja, se o objeto da classe que contém a referência deixar de existir, o objeto por ele referenciado também deixará de existir. Uma agregação pode conter multiplicidade ao lado da classe que está contida em outra. A simbologia 1.. * significa: uma classe pode estar agregada a muitas classes.
	Representa herança entre classes.

A linguagem UML estabelece vários outros símbolos para o diagrama de classes. Nessa tabela foram apresentados somente aqueles utilizados no decorrer deste documento.

2.4.2 Diagrama de interação

A interação entre objetos em um sistema orientado a objetos é feita por meio de mensagens. "Uma interação é um comportamento que abrange uma série de mensagens trocadas entre um conjunto de objetos em um contexto a fim de atingir um propósito" [6].

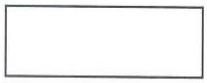
O diagrama de interação modela o aspecto dinâmico do relacionamento entre objetos. Existem alguns formatos propostos na UML para

o diagrama de interação. O escolhido nesta dissertação representa a troca de mensagens entre objetos de acordo com a passagem do tempo.

Nesse diagrama, os objetos são colocados alinhados na parte superior do desenho. Cada um é representado por um retângulo com o objeto e/ou o nome da classe sublinhados. Uma linha vertical tracejada, chamada *linha do tempo* do objeto, indica a sua execução durante a seqüência (isto é, mensagem enviadas ou recebidas). A comunicação entre os objetos é representada como linhas de mensagem horizontais entre as linhas do tempo. Lê-se um diagrama de seqüência de cima para baixo a fim de ver a troca de mensagens acontecendo durante a execução do programa.

Assim que um objeto recebe uma mensagem, é desenhado um retângulo sobre a linha do tempo. Enquanto o retângulo existir, o objeto estará ativo, seja executando seus próprios métodos, seja aguardando a resposta a uma mensagem que tenha sido enviada.

TABELA 2 – SÍMBOLOS UTILIZADOS EM UM DIAGRAMA DE INTERAÇÃO

Símbolo	Significado
	Representa um objeto. O nome da classe à qual o objeto pertence aparecerá sempre depois da descrição do próprio objeto, sendo que esta última é opcional. Nos diagramas utilizados nesse documento aparecem somente os nomes das classes às quais os objetos pertencem.
	Representa a linha do tempo de um objeto. Aparece sempre no sentido vertical.
	Quando um objeto recebe uma mensagem passa a estar ativo. Esse estado é representado por um retângulo sobre a linha do tempo.
	Representa uma mensagem enviada de um objeto a outro. Sobre a linha aparece o nome do método invocado e, entre parênteses, o tipo de seus parâmetros. Antes do nome do método pode aparecer o símbolo * (asterisco). Ele significa que a chamada a este método é iterativa. Um objeto pode enviar uma mensagem a si mesmo.

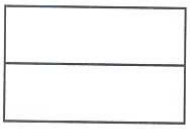
A linguagem UML estabelece vários outros símbolos para o diagrama de interação. Nessa tabela foram apresentados somente aqueles utilizados no decorrer deste documento.

2.4.3 Diagrama de objetos

Em um diagrama de classes são apresentados os relacionamentos possíveis entre todas as classes. Durante a execução de um sistema que tenha sido construído com base na utilização de objetos, entretanto, em um dado momento, alguns objetos estarão instanciados e relacionados a outros.

Um diagrama de objetos representa essa visão instantânea dos objetos em determinado momento na execução de um programa. É, portanto, uma representação estática do relacionamento entre algumas instâncias das classes representadas no diagrama de classes.

TABELA 3 – SÍMBOLOS UTILIZADOS EM UM DIAGRAMA DE OBJETOS

Símbolo	Significado
	Representa um objeto. Na parte superior aparece a descrição do objeto (opcional) separada do nome da classe (obrigatório) pelo sinal de dois pontos. A parte inferior é opcional. Ela mostra o conteúdo dos atributos do objeto, mas apenas aqueles que sejam relevantes para a compreensão dos modelos.
	Representa uma associação entre objetos.

A linguagem UML estabelece outros símbolos para o diagrama de objetos. Nessa tabela foram apresentados somente aqueles utilizados no decorrer deste documento.

Neste capítulo foram apresentadas algumas das metodologias utilizadas no desenvolvimento desta dissertação. O objetivo era o de revisar alguns conceitos que serão assumidos como conhecidos na descrição do protótipo desenvolvido.

3 DESCRIÇÃO DO PROTÓTIPO

Nesse capítulo será descrito o protótipo desenvolvido com o objetivo de mostrar que é possível reconhecer automaticamente a estrutura de um documento HTML e armazenar o seu conteúdo num banco de dados relacional temporal. Em primeiro lugar, será apresentado o processo como um todo, sem descrição pormenorizada de como são obtidos os resultados intermediários. Depois, serão descritos todos os passos empregados na inferência da estrutura do documento, na extração do conteúdo e na tradução dessa estrutura em comandos SQL necessários à criação do banco de dados e à inclusão de informações nas relações geradas.

3.1 INTRODUÇÃO

O objetivo deste trabalho é mostrar que é possível a inferência da estrutura de um documento HTML e posterior extração do conteúdo desse documento para armazenamento em uma base de dados relacional temporal. Para demonstrar a viabilidade dessa proposta foi desenvolvido um protótipo que efetivamente realiza essas tarefas. Antes de descrever como foi feita essa implementação, entretanto, será feita uma breve descrição do procedimento como um todo, a fim de que se torne mais claro o conteúdo apresentado posteriormente.

Em primeiro lugar, é necessária a identificação de um documento HTML *d1* que sofra atualizações com periodicidade relevante (diariamente, por exemplo). A partir desse ponto, o processamento mostrado na **Figura 2** pode ser realizado.

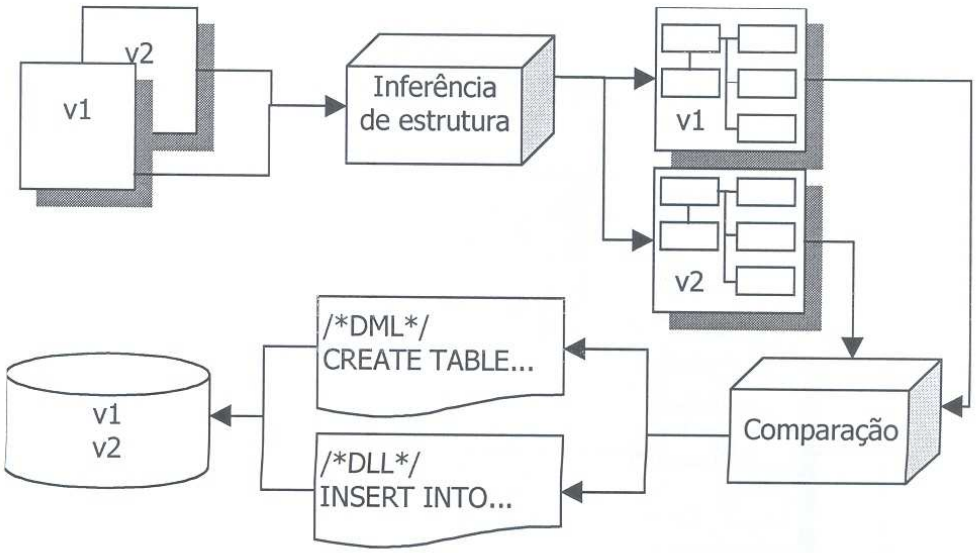


FIGURA 2 – CRIAÇÃO A BASE DE DADOS

Duas versões *v1* e *v2* temporalmente diferentes de *d1* são salvas. As duas são submetidas a um processamento que infere a estrutura subjacente aos dois documentos e que gera um modelo de objetos instanciados os quais contêm, além das ligações que permitem recuperar a estrutura do documento, os conteúdos existentes nos documentos originais. Esse módulo de inferência, portanto, também é responsável pela extração do conteúdo do documento.

Em seguida, os dois modelos de objetos são submetidos a um módulo que os compara estruturalmente. Nesse momento, o conteúdo que eles por acaso possuam não é analisado, já que interessa saber somente se os dois documentos gerariam o mesmo esquema de banco de dados. Caso isso se verifique, a estrutura de objetos de *v1* é utilizada para criação de um *script s1* composto dos comandos DDL necessários para descrever o esquema de um banco de dados. Na seqüência, *v1* e *v2* são submetidos a um processamento que gera um outro *script s2* composto dos comandos DML necessários para a inclusão de tuplas no banco de dados gerado pela execução de *s1*. *s2* também é executado a fim de que a base possua efetivamente o conteúdo de *v1* e *v2*.

Depois que a base de dados tenha sido criada, um outro tipo de processamento é realizado. Ele é demonstrado na **Figura 3**.

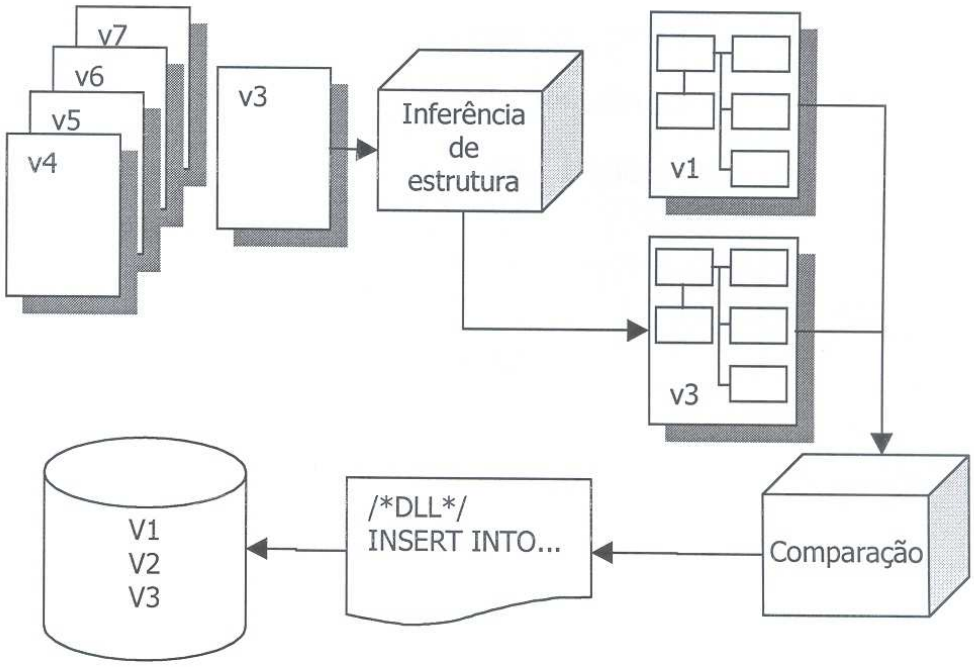


FIGURA 3 – ATUALIZAÇÃO DA BASE DE DADOS

Sucessivas versões de *d1* utilizado na criação da base de dados devem ser obtidas (*v3, v4, v5, v6, v7, ...*). Tais versões, uma a uma, são submetidas ao módulo que infere a estrutura e extrai o conteúdo do documento, criando o modelo de objetos instanciados. Esse modelo é processado pelo módulo de comparação, juntamente com o modelo de *v1* obtido na execução anterior o qual deve ter sido mantido. Como a geração do esquema de banco de dados foi feita com base em *v1*, é ele quem deve ser utilizado no momento da comparação para verificar se não houve alteração estrutural em uma das versões que inviabilize a inclusão de seu conteúdo na base de dados já criada.

No caso de a comparação verificar que a estrutura do documento ainda permanece fiel àquela obtida com *v1*, o *script* de banco de dados contendo os comandos DML necessários para a inclusão do conteúdo de *v3* na base de dados é gerado e executado.

O mesmo processamento acontece com as sucessivas versões de

d1, na periodicidade adequada para refletir alterações de conteúdo significativas.

Esta seção descreveu como o processo ocorre. A próxima, descreverá como efetivamente é feita a inferência da estrutura, extração do conteúdo e criação dos *scripts* de banco de dados mencionados anteriormente.

3.2 DESCRIÇÃO DA IMPLEMENTAÇÃO

Em primeiro lugar, é necessário recuperar os objetivos desse protótipo:

(i) gravar em um arquivo texto um *script* de banco de dados que conterá todos os comandos SQL necessários para criar uma base de dados, bem como os comandos SQL que incluirão os dados relativos às páginas que foram utilizadas para a criação do esquema do banco de dados, e

(ii) gravar em um arquivo texto um *script* de banco de dados que conterá os comandos SQL necessários para incluir novas informações no banco de dados gerado pelo uso anterior do protótipo.

Para realizar essas atividades, o protótipo passa por três fases:

(i) inferência da estrutura e extração de conteúdo de cada uma das versões de um documento HTML,

(ii) comparação das estruturas geradas, e

(iii) caso as estruturas sejam compatíveis, criação do *script* de banco de dados.

Também foi desenvolvido um utilitário que executa o *script* gerado e, efetivamente, cria a base de dados.

Cada uma das fases apresentadas será descrita a seguir.

3.3 INFERÊNCIA DA ESTRUTURA E EXTRAÇÃO DO CONTEÚDO DE UM DOCUMENTO HTML

Com o objetivo de identificar a estrutura de cada versão de uma página HTML, foi necessário, em primeiro lugar, selecionar os elementos relevantes para alimentação do banco de dados. Com base no estudo da linguagem HTML, verificou-se que um documento pode conter as seguintes partes:

- **Definição de estrutura**: é o local no qual existem as informações sobre o próprio documento. São metainformações as quais são geralmente ignoradas pelo navegador e invisíveis para o visualizador mas que são utilizadas pelos mecanismos de busca com o objetivo de "entender, categorizar, rotular e acessar informações de documentos Web." [28] O único elemento de estrutura que é considerado por este módulo é o que especifica a versão da HTML usada no documento (<!DOCTYPE>). Neste trabalho, foi utilizada a versão 4 da linguagem HTML e o protótipo recusa o documento que não pertencer a essa versão, ou a uma versão anterior a essa.
- **Descrição de molduras**: uma página *web* pode ser, na realidade, uma composição de várias outras páginas. O uso de molduras (*frames*), possibilita que uma única página *web* apresente o conteúdo de mais de um documento HTML. O documento original, nesse caso, possui somente a indicação do local em que deve buscar os outros documentos que serão apresentados. Este protótipo ainda não considera o uso de *frames* na criação da estrutura do documento HTML que deve ser reconhecido, mas a implementação desse recurso é relativamente simples e poderá fazer parte de desenvolvimentos futuros da ferramenta.

- **Corpo do documento**: é o local no qual estão os dados propriamente ditos. Tudo o que é gravado em banco de dados reside no corpo do documento, o qual é delimitado pelos rótulos <BODY> e </BODY>. A quase totalidade do processamento feito pelo protótipo reside na análise do corpo de um documento HTML.

É importante ressaltar que, embora tenha sido feito um estudo aprofundado da linguagem HTML, este protótipo não verifica se o código HTML é válido. Ele assume que a página está construída corretamente. Códigos inválidos terão resultados imprevisíveis. Por exemplo, o elemento <CITE> não permite elementos <P> internos e, ainda, pressupõe um rótulo final. Se este não existir ou se houver elementos inválidos no seu interior, não é possível prever o que haverá como resultado final da avaliação do código HTML.

Com base na identificação da composição de um documento HTML, foi criado um modelo de objetos apresentados na **Figura 4**. A representação da página na forma de objetos foi escolhida por questões de implementação. Inicialmente a página seria representada por meio de outra estrutura de dados, mais especificamente uma lista encadeada. Entretanto, o caráter recursivo da linguagem HTML tornava extremamente complexa essa forma de implementação. Por esse motivo foi adotada uma programação orientada a objetos com a respectiva representação da estrutura de uma página por meio dessa tecnologia.

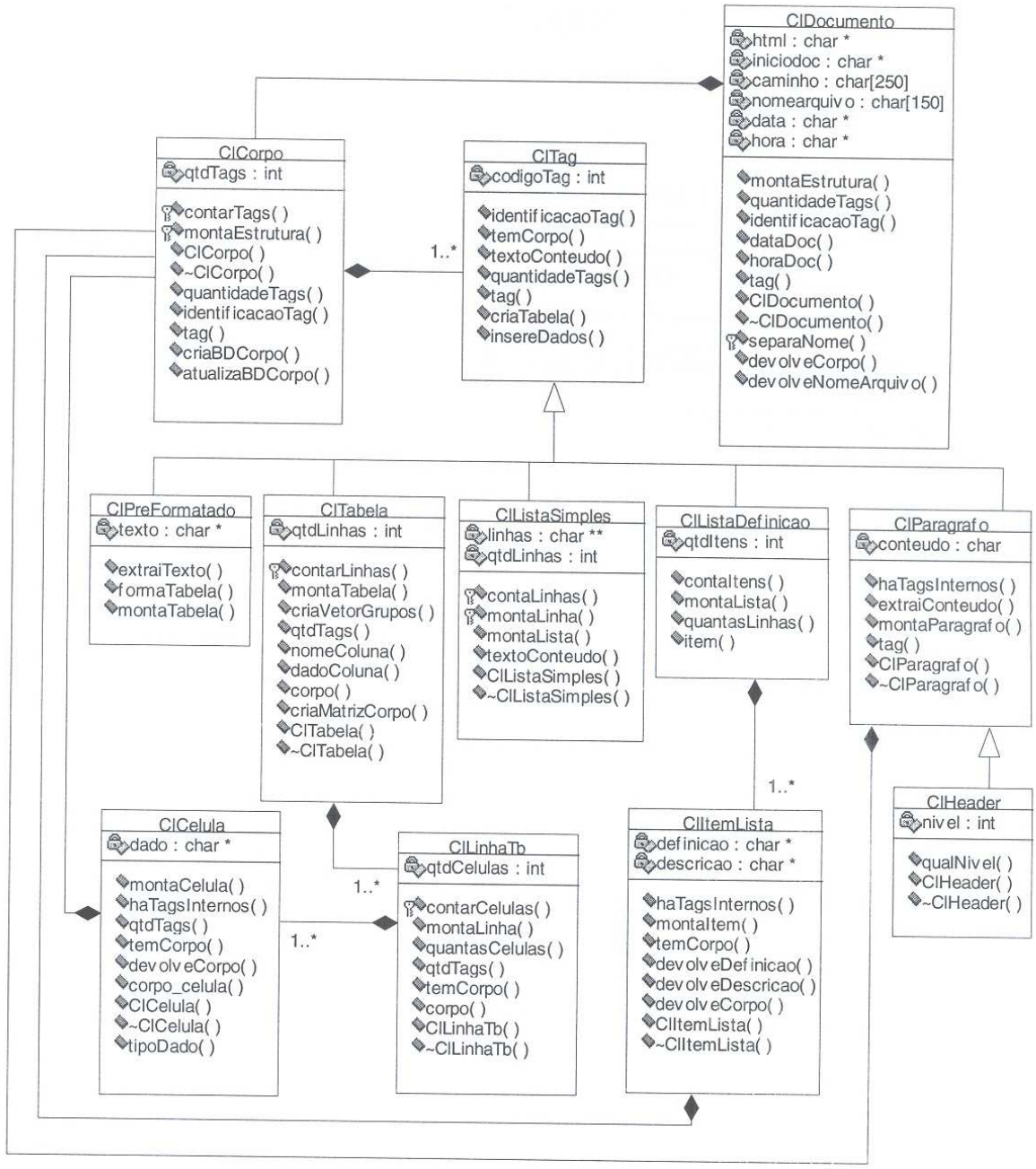


FIGURA 4 – MODELO DE OBJETOS QUE REPRESENTA UM DOCUMENTO HTML

Um documento é composto por um corpo. Este corpo, por sua vez, é composto de uma lista de rótulos. Os rótulos podem ser de cinco tipos diferentes: parágrafo, tabela, texto pré-formatado, lista simples e lista de definição. Alguns desses rótulos podem conter um corpo, como é o caso de um parágrafo. Isso torna recursiva a estrutura de objetos e a implementação do protótipo.

Os elementos considerados na implementação estão listados na **Tabela 4**. Na terceira coluna, está indicada a classe à qual pertencerá o objeto (ou atributo) criado sempre que o rótulo for encontrado no código HTML.

TABELA 4 – ELEMENTOS CONSIDERADOS NA INFERÊNCIA DA ESTRUTURA DE UMA PÁGINA

Elemento	Significado	Classe
<BLOCKQUOTE> </BLOCKQUOTE>	Define um texto contido em uma citação longa.	*ClParagrafo*
<CITE> </CITE>	Define uma citação.	*ClParagrafo*
<CODE> </CODE>	Define um texto que representa um código de programa de computador.	*ClParagrafo*
<DD>	Define o texto de descrição de um termo em uma lista de definições.	Atributo *descrição* da classe *ClItemLista*
<DIR> </DIR>	Define uma lista.	*ClListaSimples*
<DIV> </DIV>	Cria uma divisão lógica no documento	*ClParagrafo*
<DL> <DL>	Define uma lista de definições. Cada linha de uma lista de definições é composta de dois elementos: um termo e a definição desse termo.	*ClListaDefinicao*
<DT>	Define um termo de uma lista de definições.	Atributo *definição* na classe *ClItemLista*
<H*> </H*>	Cabeçalhos. O * representa o nível do cabeçalho e pode ser substituído pelos algarismos de 1 a 6.	*ClHeader*
<LI>	Define o início de uma linha em uma lista ordenada ou desordenada.	Comporá o vetor linhas da classe *ClListaSimples*
<MENU> </MENU>	Define uma lista.	*ClListaSimples*
<OL> </OL>	Define uma lista ordenada. As listas ordenadas possuem linhas numeradas.	*ClListaSimples*
<P> </P>	Define um parágrafo de texto.	*ClParagrafo*
<PRE> </PRE>	Define um texto que deve ser apresentado exatamente como foi inserido. Geralmente esse texto é mostrado em fonte monoespaçada e possui todos os espaços e quebras de linha que foram incluídas originalmente. O texto pré-formatado pode apresentar informações de forma tabular. O protótipo que foi desenvolvido avalia o conteúdo do texto pré-formatado e reconhece uma tabela, transformando o código HTML original nos rótulos correspondentes, a fim de que o banco de dados seja criado levando em consideração esse fato.	*ClPreFormatado*
<SAMP> </SAMP>	Define um texto que representa a saída de um programa de computador.	*ClParagrafo*
<TABLE> </TABLE>	Define uma tabela.	*ClTabela*
<TD> </TD>	Define uma célula de uma tabela.	*ClCelula*
<TR> </TR>	Define uma linha de uma tabela.	*ClLinhaTabela*
<UL> </UL>	Define uma lista desordenada. As listas desordenadas possuem marcadores no início de suas linhas .	*ClListaSimples*

Alguns desses elementos estão depreciados, isto é, a recomendação HTML sugere que eles não sejam usados e que, em seu lugar, sejam

empregados outros elementos mais adequados. É o caso dos rótulos <DIR> e <MENU>. A especificação HTML sugere que eles sejam evitados e, no seu lugar, utilize-se o rótulo <UL>[29]. Entretanto, o protótipo não tem a intenção de fazer juízos de valor sobre a construção da página, ele apenas pretende reconhecer a estrutura que já está criada. Portanto, como esses elementos podem aparecer em um documento, eles são analisados durante a execução do programa.

A escolha dos elementos a serem considerados se fez com base na avaliação daqueles que efetivamente conteriam informações que pudessem ser relevantes para a criação de um banco de dados. Por esse motivo os elementos de formatação foram descartados e elementos como os de citação e de inclusão de código foram considerados. Acreditou-se que esses últimos poderiam ter informações as quais interessariam ao usuário de um futuro sistema que utilize essa tecnologia.

Com base nas informações apresentadas, pode-se descrever como é feito o reconhecimento da estrutura de um documento HTML. O diagrama de seqüência da **Figura 5** demonstra essa tarefa:

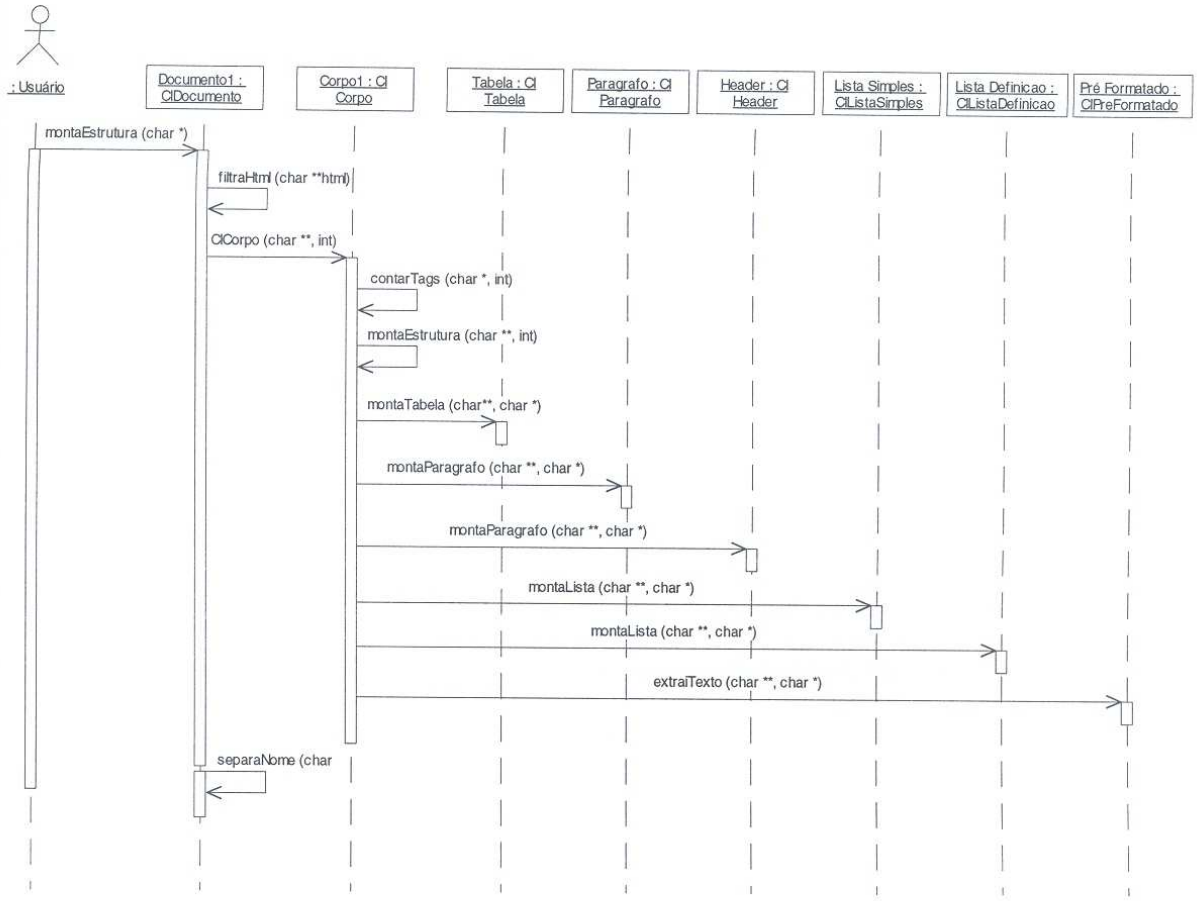


FIGURA 5 – DIAGRAMA DE SEQÜÊNCIA PARA O RECONHECIMENTO DA ESTRUTURA DE UM DOCUMENTO

O arquivo HTML é aberto e seu código é lido. Informações relevantes do documento como data, hora[2] e nome do arquivo são retiradas e, em seguida, o código HTML é filtrado. Isso significa dizer que somente os rótulos relevantes são deixados no texto do documento. Todos os outros são retirados. Em seguida, o corpo do documento é criado.

O objeto *corpo* faz uma pré-análise do código HTML a fim de identificar quantos são os rótulos relevantes. O objetivo dessa análise é saber qual será o tamanho do vetor responsável por criar a agregação do tipo 1..* entre as classes *ClCorpo* e *ClTag*. Nessa análise não são considerados os rótulos que fazem parte do corpo de um rótulo em particular. Por exemplo, com base no código HTML da **Figura 6**:

[2] São essas informações que garantirão o caráter temporal da base de dados

```
<html>
<body>
<p>
    <cite>Citação 1</cite>
    <cite>Citação 2</cite>
</p>
<p>
    <cite>Citação 3</cite>
    <cite>Citação 4</cite>
</p>
</body>
</html>
```

FIGURA 6 – CÓDIGO HTML DE DOCUMENTO QUE POSSUI RÓTULOS COM CORPO

Em primeira análise, observando a lista dos rótulos que são considerados no reconhecimento da estrutura de um documento HTML, esse código possuiria um total de 6 rótulos: dois parágrafos do tipo <P> e quatro parágrafos do tipo <CITE>. Entretanto, os parágrafos do tipo <CITE> estão dentro de parágrafos do tipo <P>, fazem parte do corpo destes, e não são considerados no momento da contagem de quantos rótulos existem no documento. No exemplo apresentado anteriormente, o documento possuirá somente dois rótulos: dois parágrafos do tipo <P>.

Caso o documento possua rótulos relevantes, ou seja, um daqueles pertencentes à **Tabela 4**, o seu corpo será montado por meio da execução do método *montaEstrutura*. Esse método executa iterativamente uma rotina de *parsing*, a qual reconhece um *token* e seu respectivo tipo. Caso o *token* represente um rótulo inicial que esteja entre aqueles da **Tabela 4**, o método que faz o reconhecimento do objeto correspondente será executado. Isso será feito até que o código HTML do documento seja totalmente reconhecido.

A seguir serão explicados os procedimentos utilizados para reconhecimento de cada um dos rótulos que podem fazer parte de um documento.

3.3.1 Reconhecimento de uma tabela

O processamento das tabelas do código HTML é o mais sofisticado nesse protótipo. Em todos os momentos em que houver especificação do tratamento dado a cada classe, será possível perceber que aquele dispensado às tabelas será o mais detalhado. Os motivos são práticos:

(i) verificou-se que, em páginas reais da Internet, as informações aparecem em grande parte dos documentos pesquisados no formato tabular, esteja esse formato visível ou não para o usuário. Isso acontece, principalmente, com as informações relevantes para futuro armazenamento na base de dados;

(ii) a linguagem HTML possibilita especial complexidade para a construção de tabelas e tentou-se cobrir a maior parte de situações passíveis de serem encontradas em páginas reais;

(iii) por fim, como objetivo colateral deste trabalho, pretende-se disponibilizar as bases de dados geradas para futuro tratamento e adaptação a aplicações de *data mining*, e informações dispostas em formato tabular têm, a princípio, a maior possibilidade de serem interessantes para esse tipo de produto.

Em relação ao reconhecimento da estrutura de uma tabela, o diagrama de seqüência da **Figura 7** demonstra tal processo.

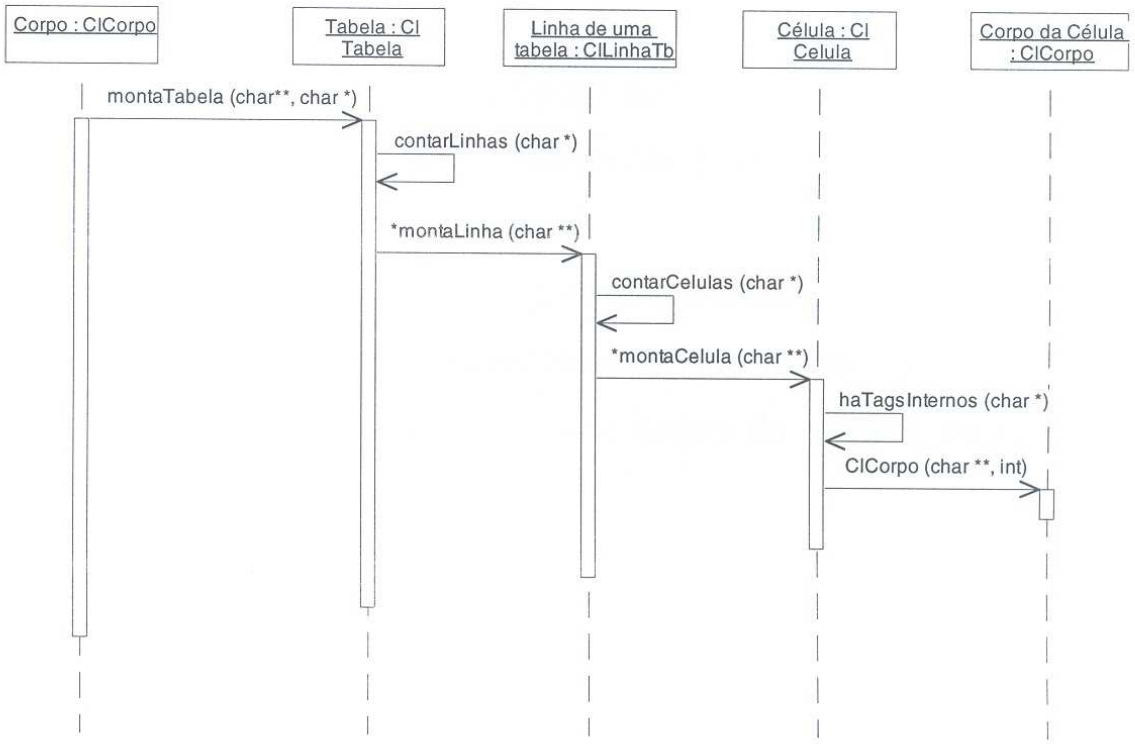


FIGURA 7 – DIAGRAMA DE SEQÜÊNCIA PARA O RECONHECIMENTO DE UMA TABELA

Um objeto que pertença à classe *ClTabela* precisa saber quantas linhas possui a fim de que seja determinado o tamanho do vetor que implementa a agregação do tipo 1..* que existe entre as classes *ClTabela* e *ClLinhaTb*. Em seguida, o método *montaLinha* da classe *ClLinhaTb* será executado iterativamente para cada linha da tabela.

Da mesma forma que acontece com uma tabela, a montagem de uma linha conta, em primeiro lugar, quantas células existem a fim de que seja determinado o tamanho do vetor que implementa a agregação do tipo 1..* que existe entre as classes *ClLinhaTb* e *ClCelula*. Depois, o método *montaCelula* é executado iterativamente, tantas vezes quantas forem as células da linha.

Por fim, dentro do método *montaCelula* é feita a verificação da existência, ou não, de um corpo para a célula. Uma célula possui um corpo quando tem, dentro dela, um ou mais elementos pertencentes à **Tabela 4**. Caso isso ocorra, o método que cria um corpo será executado. Isso estabelece o caráter recursivo da criação do corpo de um documento, já que um corpo pode ser criado antes mesmo que o método de reconhecimento do

corpo de um documento como um todo seja concluído. É por esse motivo que o método construtor da classe *ClCorpo* possui o parâmetro **hierarquia**. É esse parâmetro que indica para o método até onde ir no código HTML que ele recebe para analisar.

Possuindo ou não elementos internos, o método *montaCelula* segue concatenando todas as informações textuais que fizerem parte do conteúdo da célula. Essas informações serão o conteúdo do atributo **dado** do objeto que pertence à classe *ClCelula*.

3.3.2 Reconhecimento de um parágrafo

O diagrama de seqüência da **Figura 8** demonstra o processo de reconhecimento de um parágrafo:

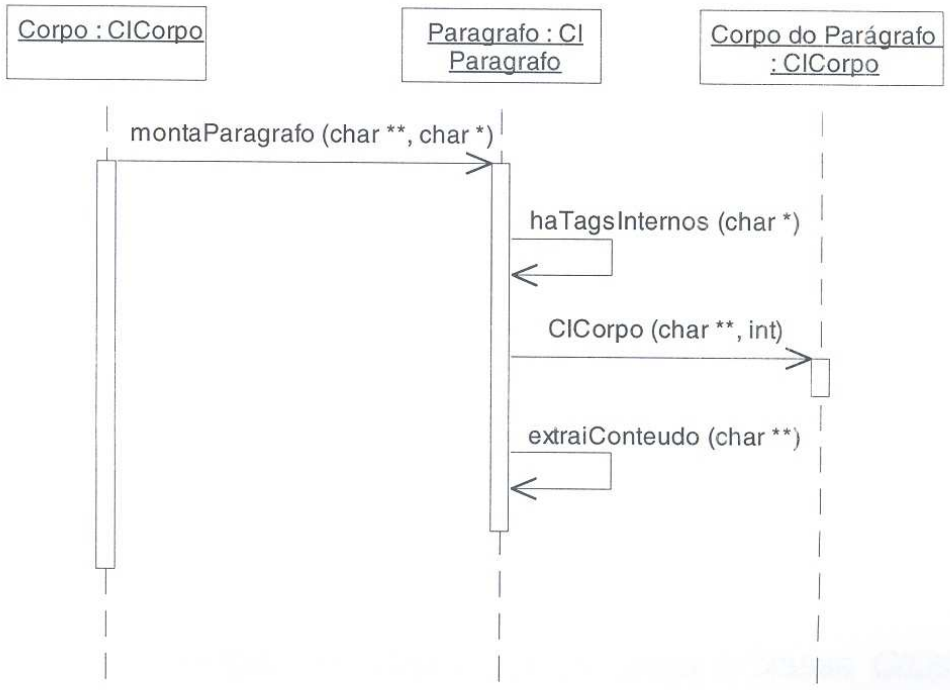


FIGURA 8 – DIAGRAMA DE SEQÜÊNCIA PARA O RECONHECIMENTO DE UM PARÁGRAFO

O método que monta um parágrafo verifica se o mesmo tem rótulos internos, o que caracteriza o fato de o parágrafo possuir um corpo. Caso isso ocorra, o corpo do parágrafo será montado. Novamente, ocorre o processo

recursivo de montagem do corpo, já que isso acontecerá antes que o método que reconhece o corpo do próprio documento tenha sido encerrado.

Caso o parágrafo não possua corpo, seu conteúdo textual será retirado e será o valor do atributo **conteudo**[3] da classe *ClParagrafo*. Vale lembrar que os seguintes rótulos constituirão objetos dessa classe: <DIV>, <CODE>, <SAMP>, <BLOCKQUOTE> e <P>.

Um cabeçalho é reconhecido da mesma forma que um parágrafo. A diferença está no método construtor da classe *ClHeader*, o qual identifica o conteúdo do atributo **nivel** com base no próprio rótulo (<H1>, <H2>, etc.).

3.3.3 Reconhecimento de uma lista simples

O diagrama de seqüência da **Figura 9** demonstra o processo de reconhecimento de uma lista simples.

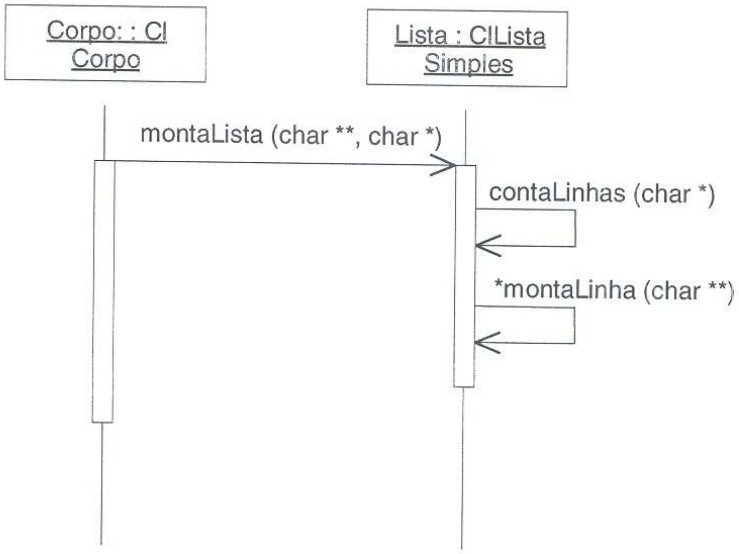


FIGURA 9 – DIAGRAMA DE SEQÜÊNCIA PARA O RECONHECIMENTO DE UMA LISTA SIMPLES

Em primeiro lugar, um objeto que pertença à classe *ClListaSimples* precisa saber quantas linhas possui, a fim de que a quantidade de elementos do vetor **linhas** (atributo de *ClListaSimples*) possa ser determinada. Em

[3] Nomes de atributos e métodos serão referenciados sem acentuação com o objetivo de manter a coerência com os nomes utilizados na implementação, apresentados nos diversos modelos e descritos no anexo.

seguida, o método *montaLinha* é executado iterativamente, tantas vezes quantas forem as linhas da lista. Esse método tem o objetivo de compor o conteúdo de cada um dos elementos do vetor **linhas**. Uma linha de uma lista simples não pode conter internamente nenhum dos rótulos que geram um corpo.

Os seguintes rótulos são considerados listas simples: <OL>, <UL>, <DIR> e <MENU>.

3.3.4 Reconhecimento de uma lista de definição

O diagrama de seqüência da **Figura 10** demonstra o processo de reconhecimento de uma lista de definição.

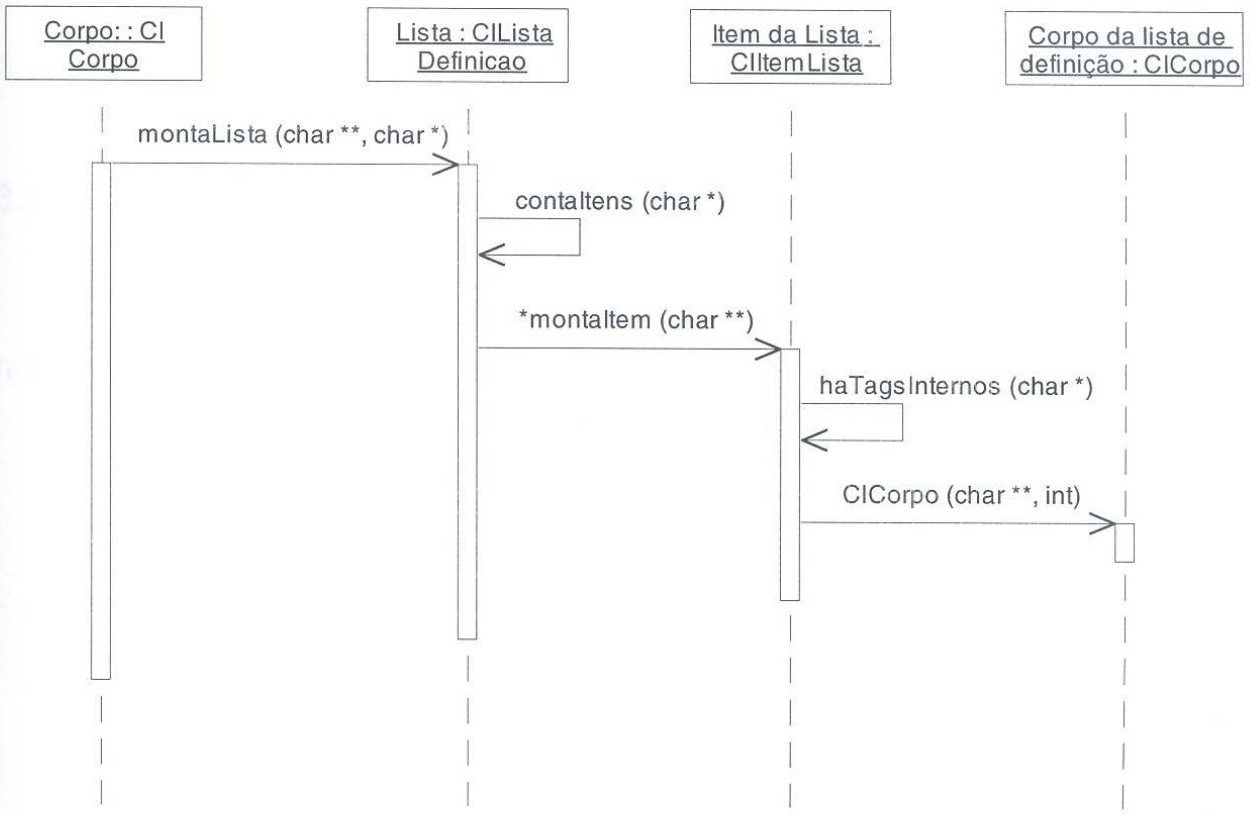


FIGURA 10 – DIAGRAMA DE SEQÜÊNCIA PARA O RECONHECIMENTO DE UMA LISTA DE DEFINIÇÃO

Em primeiro lugar são contados quantos itens uma lista de definição contém. Isto é feito para que se possa determinar o tamanho do vetor **itens**, o qual implementa a agregação 1..* entre as classes *CIListaDefinicao* e *CIItemLista*. Depois de calcular a quantidade de itens de uma lista, o método

que efetivamente os cria é executado iterativamente, tantas vezes quantos forem os itens.

Um item de uma lista de definição é sempre composto de uma definição (um termo simples) juntamente com a descrição dessa definição. Como dito anteriormente, a definição é sempre simples, isto é, não possui elementos internos. O mesmo não acontece com a descrição. Portanto, a montagem de um termo constitui-se de dois passos: a extração do conteúdo da definição (atributo **definicao** da classe *ClItemLista*) e, em seguida, uma verificação que indica se a descrição possui ou não um corpo. Caso o possua, ele será montado, caracterizando novo processo recursivo.

Se esse corpo não existir, o conteúdo da descrição é extraído do código HTML e atribuído a **descricao**.

O rótulo que caracteriza uma lista de definição é <DL>.

3.3.5 Reconhecimento de um texto pré-formatado

O diagrama de seqüência da **Figura 11** demonstra o processo de reconhecimento de um texto pré-formatado:

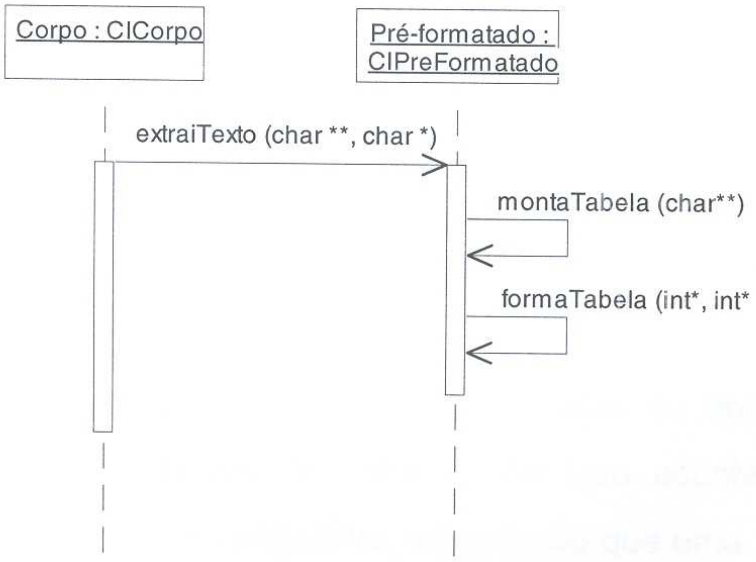


FIGURA 11 – DIAGRAMA DE SEQÜÊNCIA PARA O RECONHECIMENTO DE UM TEXTO PRÉ-FORMATADO

Um texto pré-formatado, definido pelo rótulo <PRE>, tem o objetivo de apresentar o texto de seu conteúdo tal qual foi digitado pelo criador do código HTML. Ele não pode possuir um corpo interno, mas, em muitos documentos, pode ter o objetivo de apresentar informações no formato tabular.

Nesse protótipo, procurou-se reconhecer esse fato, modificando o código HTML original, a fim de que passe a conter não um texto pré-formatado, o qual seria incluído no banco de dados como um texto único, mas sim como uma tabela, já que, provavelmente, era essa a intenção do autor.

O método *montaTabela* em primeiro lugar verifica se o texto forma ou não uma tabela. Caso isso seja verdade, o código HTML é modificado a fim de que sejam incluídos os rótulos que a representam. Esse código é retornado ao método que monta o corpo de um documento, com o objetivo de que sejam executados os métodos relativos a uma tabela.

Quem identifica se um texto representa ou não uma tabela é o método *formaTabela*. Esse método faz o seguinte processamento:

(i) Conta quantas linhas possui o texto, identificando a quantidade de quebras de linhas.

(ii) Cria um vetor de inteiros com tantos elementos quantas forem as linhas.

(iii) Para cada linha, identifica quantas colunas existem, contando quantos tokens separados por espaços existem. Essa quantidade é armazenada na posição relativa à linha no vetor criado no passo anterior.

(iv) Percorre esse vetor verificando se todas as linhas possuem a mesma quantidade de colunas. Se isso acontecer, o método retorna um valor verdadeiro, informando que uma tabela pode ser criada.

(v) Caso as linhas não possuam todas a mesma quantidade de

colunas, o método verifica se somente a primeira linha possui uma coluna a menos do que as demais. Nesse caso, também pode ser formada uma tabela, a qual possui títulos de colunas e de linhas. Caso isso ocorra, um espaço incondicional representado pela entidade de caractere * * [28][29] é inserido na primeira coluna da primeira linha. Isso é feito para que o método que reconhece a tabela não ignore a célula criando-a efetivamente, mas com conteúdo vazio. O método retorna um valor informando que uma tabela pode ser criada.

(vi) Se nenhuma das duas situações anteriores for verificada, o método retorna um valor informando que uma tabela não pode ser criada. Nesse caso, todo o conteúdo do texto pré-formatado será acrescentado ao atributo **texto** da classe *ClPreFormatado*.

A página apresentada na **Figura 12** possui três textos pré-formatados. Os dois primeiros podem ser convertidos em tabela, e o serão pelo protótipo. O terceiro, não.

```
        Coluna1    Coluna2    Coluna3
Linha1      2          5          7
Linha2      5          1          1.4
Linha3      0          4          8



Coluna1    Coluna2    Coluna3
   2          5          7
   5          1          1.4
   0          4          8



Coluna1                  Coluna3
   2          5          7
   5          1          1 e 4
              4          8
```

FIGURA 12 – EXEMPLO DE PÁGINA QUE CONTÉM TEXTOS PRÉ-FORMATADOS

O código HTML que deu origem a essa página é o seguinte:

```
<html>
<body>

<pre>
       Colunal    Coluna2    Coluna3
Linhal     2          5          7
Linha2     5          1          1.4
Linha3     0          4          8
</pre>

<pre>
Colunal    Coluna2    Coluna3
   2          5          7
   5          1          1.4
   0          4          8
</pre>

<pre>
Colunal                  Coluna3
   2          5          7
   5          1          1 e 4
              4          8
</pre>

</body>
</html>
```

FIGURA 13 – CÓDIGO HTML COM EXEMPLO DE TEXTOS PRÉ-FORMATADOS

O primeiro texto tem três colunas na primeira linha e quatro colunas nas três últimas. Serão inseridos no código HTML original, uma entidade de caractere que representa um espaço incondicional além dos rótulos de tabela necessários.

O segundo texto possui três colunas em todas as linhas. Novamente, serão inseridos no código HTML os rótulos que representam uma tabela.

O terceiro texto não pode ser convertido em tabela porque possui diferentes quantidades de colunas em cada linha (duas na primeira, três na segunda, cinco na terceira e duas na última). Como as colunas são contadas em função dos espaços em branco entre os *tokens*, a terceira linha possuirá cinco colunas.

O código HTML alterado, depois do processamento, ficará da seguinte forma:

```
<html>
<body>
<table> <tr> <td> </td> <td>Colunal</td> <td>Coluna2</td> <td>Coluna3</td> </tr>
        <tr> <td>Linhal</td> <td>2</td> <td>5</td> <td>7</td> </tr>
        <tr> <td>Linha2</td> <td>5</td> <td>1</td> <td>1.4</td> </tr>
        <tr> <td>Linha3</td> <td>0</td> <td>4</td> <td>8</td> </tr>
</table>

<table> <tr> <td>Colunal</td> <td>Coluna2</td> <td>Coluna3</td> </tr>
        <tr> <td>2</td> <td>5</td> <td>7</td> </tr>
        <tr> <td>5</td> <td>1</td> <td>1.4</td> </tr>
        <tr> <td>0</td> <td>4</td> <td>8</td> </tr>
</table>

<pre>
Colunal              Coluna3
   2         5         7
   5         1        1 e 4
             4         8
</pre>
</body>
</html>
```

FIGURA 14 – CÓDIGO HTML MODIFICADO PELO PROCESSAMENTO DE TEXTOS PRÉ-FORMATADOS

Embora o reconhecimento de texto pré-formatado em apresentação tabular seja bastante limitado, não houve a intenção de fazer grande esforço para melhorar esse reconhecimento pelo fato que autores de documentos HTML atualmente se utilizam mais do recurso tabela propriamente dito do que do uso de texto pré-formatado. Entretanto, julgou-se interessante incluir no protótipo a habilidade de reconhecer situações bastante simples, as quais poderiam ser de valia para o usuário de um produto que venha a utilizar a tecnologia que foi desenvolvida.

3.3.6 Exemplo de resultado de reconhecimento de documento HTML

Com o objetivo de tornar claro como se processa o reconhecimento da estrutura de um documento HTML, será apresentado um exemplo de código que contém algumas das características tratadas pelo protótipo e discutidas anteriormente (em especial, o caráter recursivo dos rótulos). Em seguida, será mostrado um diagrama de objetos instanciados, o qual representa o que é

gerado como resultado desse processamento. Os atributos que podem aparecer nos rótulos foram eliminados, uma vez que são ignorados, a fim de que o código fique mais claro. Pelo mesmo motivo, foram mostrados somente os elementos considerados na estruturação do documento.

```
<html>
<body>

<p>Este é um parágrafo de texto normal. O próximo
contém um corpo interno:</p>

<p>
   <cite>Uma citação.</cite>
   <code> Uma linha de código</code>
   <cite> Outra citação.</cite>
</p>

<div>
   <cite>
      <samp>Uma saída dentro de uma citação, que está
           inserida em uma outra hierarquia: uma divisão.
      </samp>
   </cite>
</div>

<table>
  <tr>
    <td>Célula 1</td>
  </tr>
  <tr>
    <td>Célula 2</td>
  </tr>
  <tr>
    <td>Grupo 2.1</td>
    <td>Grupo 2.2</td>
  </tr>
  <tr>
    <td>01/01/2001</td>
    <td>5</td>
  </tr>
  <tr>
    <td>04/02/2001</td>
    <td>9</td>
  </tr>
</table>

<p>A próxima tabela tem uma tabela interna:</p>
```

```
<table>
  <tr>
    <td>Coluna 1</td>
  </tr>
  <tr>
    <td><table>
      <tr>
        <td>01/01/2001</td>
        <td>2.34</td>
      </tr>
      <tr>
        <td>02/04/2001</td>
        <td>3.56</td>
      </tr>
    </table>
    </td>
  </tr>
</table>

<p>Exemplo de lista simples:

<ul>
  <li>Primeira linha</li>
  <li>Segunda linha</li>
  <li>Terceira linha</li>
</ul>

<p>Exemplo de lista de definição:

<dl>
  <dt>Termo 1 </dt>
  <dd>Este termo é simples </dd>
  <dt>Termo 2 </dt>
  <dd> <p>Este termo tem um corpo </p>
      <cite>Uma citação</cite>
      <div>Uma divisão</div></dd>
</dl>
</body>
</html>
```

O código apresentado gera uma página com a seguinte aparência:

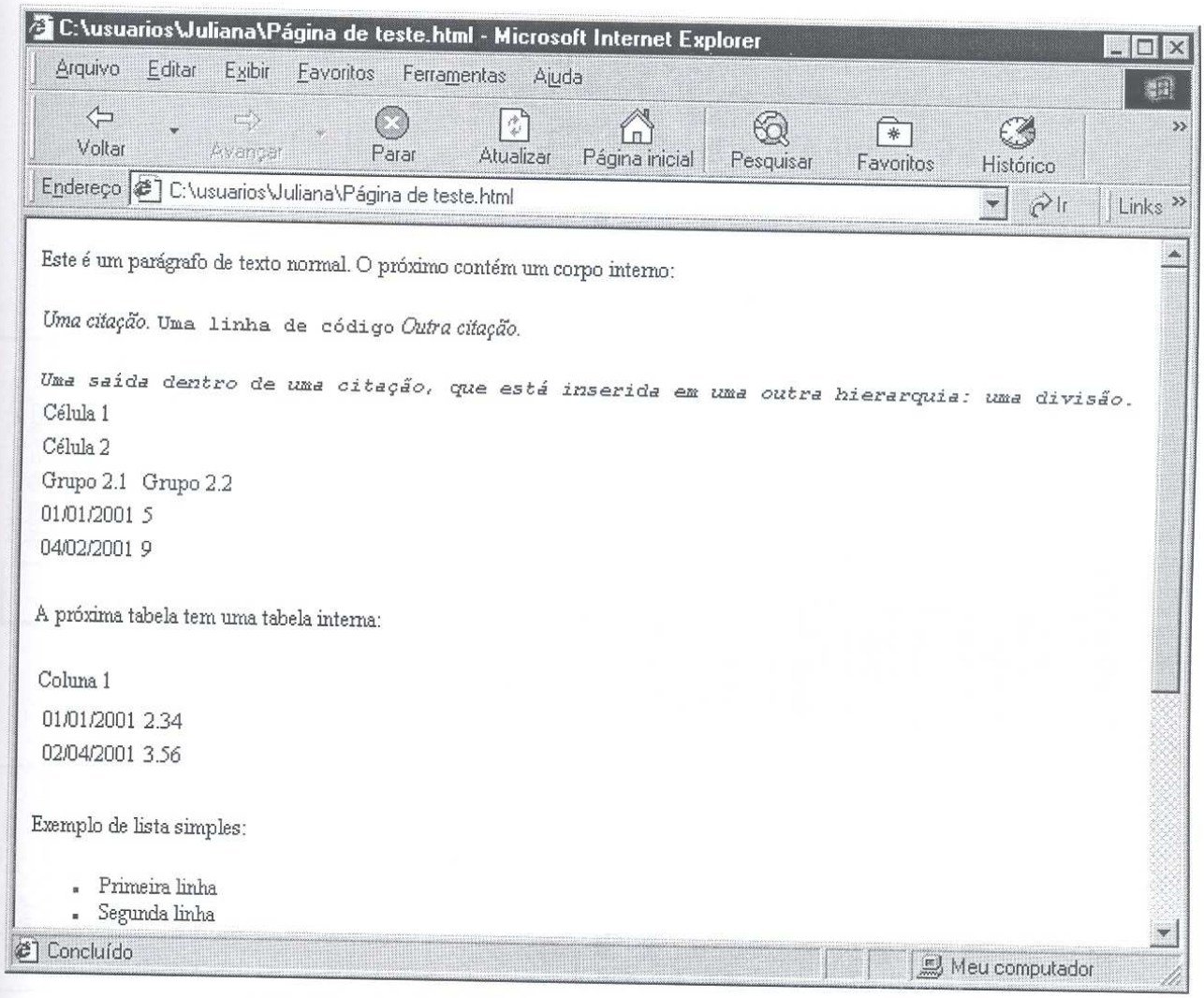


FIGURA 15 – ASPECTO DA PÁGINA UTILIZADA PARA TESTE

Com base no processamento desse documento HTML, vários objetos serão instanciados. Quais objetos, bem como as ligações entre eles estão representados no diagrama de objetos da **Figura 16**.

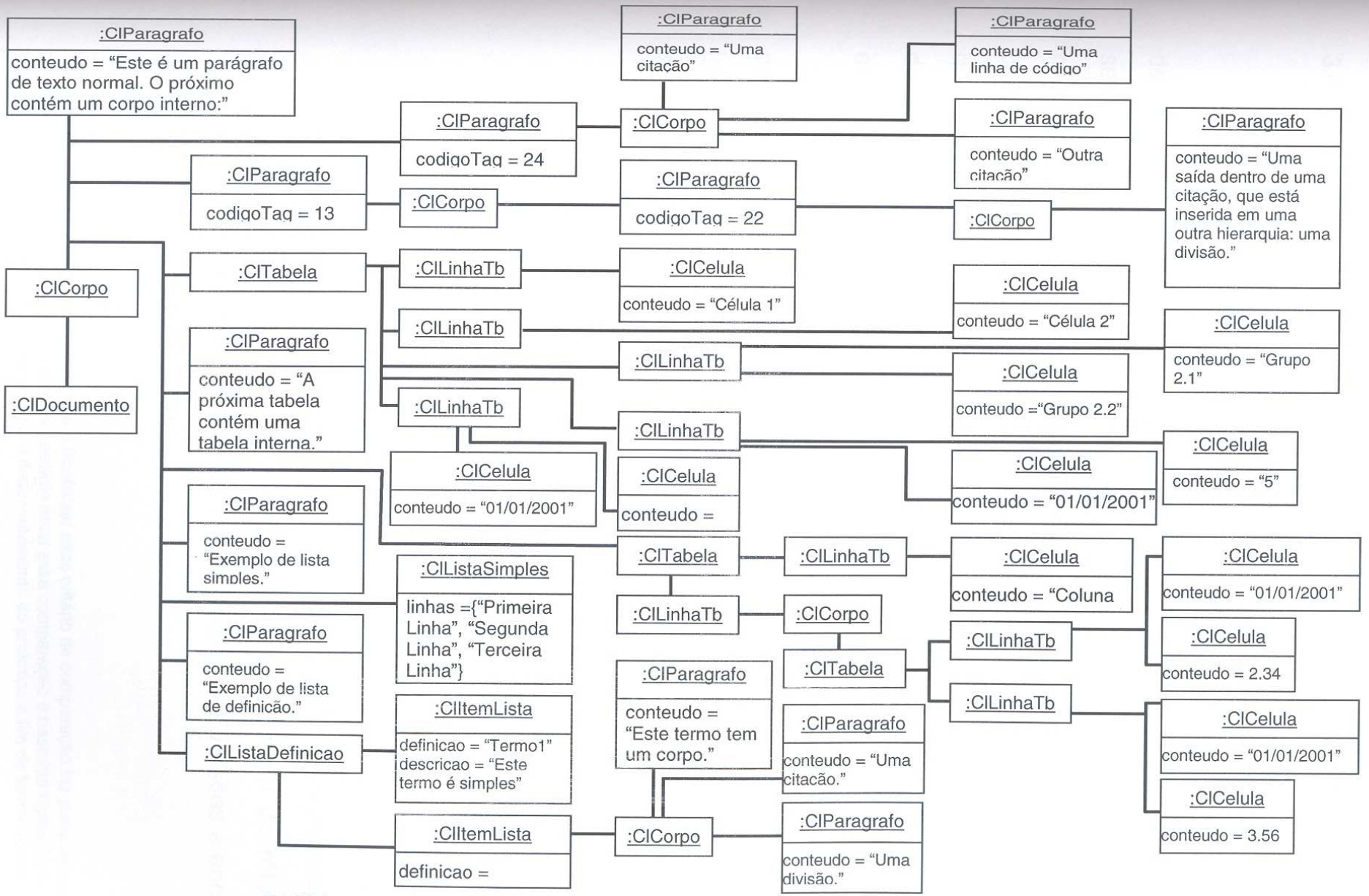


FIGURA 16 – DIAGRAMA DE OBJETOS RESULTADO DO RECONHECIMENTO DE UM DOCUMENTO HTML

3.4 COMPARAÇÃO DA ESTRUTURA DE DUAS VERSÕES DE UM DOCUMENTO

Depois que tenha sido gerada a estrutura de objetos de duas versões de um documento, estas são comparadas a fim de verificar se são semelhantes. Se, entre as versões, houver ocorrido alguma mudança estrutural, a base de dados não poderá ser gerada e/ou atualizada. Seja *V1* a primeira versão do documento e *V2* a segunda, se em *V2* aparece uma tabela que não existia em *V1*, a base de dados não poderá ser gerada porque os documentos utilizados por base são estruturalmente diferentes e gerariam esquemas de bancos de dados também diferentes.

A comparação entre as duas versões se faz por uma função que devolve um resultado *verdadeiro* se os documentos são estruturalmente iguais, ou *falso* se não o são. No momento da comparação são executados os seguintes passos:

(i) é verificada a quantidade de rótulos de cada versão. Se a quantidade de rótulos de uma versão for diferente da quantidade de rótulos da outra versão, os documentos são estruturalmente diferentes e a comparação é encerrada com resultado falso.

(ii) Caso as versões possuam a mesma quantidade de rótulos, cada dupla de rótulos (um da versão 1 e outro da versão 2), na ordem em que aparecem nos respectivos documentos é comparada. Se, em algum momento do processamento, a dupla de rótulos for estruturalmente diferente, a comparação das versões é encerrada com resultado falso.[4]

[4] A implementação de estratégias que permitam flexibilizar esse critério de comparação faz parte das propostas para desenvolvimento futuro deste trabalho. No estágio atual essa comparação é bastante rígida. Isso foi feito como primeira versão com o objetivo de agilizar o desenvolvimento do protótipo a fim de terminá-lo em tempo hábil para a verificação da validade da proposta.

(iii) Se o processamento chegar ao fim da lista de rótulos de cada versão, a comparação é encerrada com resultado verdadeiro.

Há uma função interna ao código fonte do protótipo que faz a comparação de dois rótulos. Ela o faz com base no atributo **codigoTag** que todo objeto descendente de *ClTag* possui. Da mesma forma que a função que compara versões do documento, essa função devolve um valor verdadeiro se os rótulos são estruturalmente iguais, ou falso se não o são. O seu processamento é o seguinte:

(i) se os códigos dos rótulos que estão sendo comparados forem diferentes, a comparação é encerrada com resultado falso;

(ii) se forem ambos o mesmo rótulo, os dois são comparados. Caso sejam estruturalmente iguais, a comparação é encerrada com resultado verdadeiro. Em caso contrário, a comparação é encerrada com resultado falso.

Cada rótulo possui seu modo específico de comparação. Vale lembrar que essa comparação é feita em termos estruturais e não de conteúdo. Portanto, dois parágrafos ou duas tabelas serão iguais se ambos possuírem a mesma estrutura interna. A seguir, serão explicados os modos de comparação para cada rótulo específico:

(i) **<H*>**: se os dois rótulos de cabeçalho possuírem o mesmo nível (conteúdo do atributo **nivel** da classe *ClHeader*), os rótulos são iguais, senão são diferentes.

(ii) **<P>, <CITE>, <BLOCKQUOTE>, <CODE>, <DIV> e <SAMP>**: se ambos os rótulos possuem corpo, é verificada a quantidade de rótulos de cada um dos corpos. Caso não sejam a mesma, os rótulos são diferentes. Caso sejam iguais, dois a dois os rótulos componentes dos corpos serão comparados (chamada recursiva à função de comparação de rótulos). Se em algum momento, essa

comparação não se verificar, os rótulos são diferentes. Se ela chegar com sucesso ao final dos corpos dos rótulos originais, ou seja, se todos os rótulos, tomados dois a dois, forem iguais, os rótulos pertencentes a esta categoria serão iguais.

(iii) **<MENU>, <OL>, <UL>, <DIR>, <PRE>**: esses rótulos sempre são considerados iguais estruturalmente, já que não podem apresentar corpo interno. Isso significa que duas listas são iguais, mesmo se possuírem quantidades de linhas diferentes, o que é verdade. No momento de incluir o conteúdo de listas em uma tabela de banco de dados, cada linha da lista será uma tupla na tabela correspondente. Portanto, não faz diferença a quantidade de elementos de uma lista. Os textos pré-formatados (representados pelo rótulo <PRE>) também não podem conter corpo interno, seu conteúdo será sempre textual e gerará uma única tupla no banco de dados.

(iv) **<DL>**: o atributo **descricao** da classe *ClItemLista* agregada à classe *ClListaDefinicao* pode conter um corpo. Considerando como *simples* um rótulo cujo elemento **descricao** não possui corpo e ainda, denominando o primeiro rótulo de *Lista1* e o segundo de *Lista2*, a comparação entre dois rótulos desse tipo é feita da seguinte maneira: se todos os elementos de Lista1 forem simples e se todos os elementos de Lista2 forem simples, os rótulos são estruturalmente iguais. Se houver elemento que possui corpo em Lista1 então, se na mesma posição relativa ao início da lista em Lista2 houver também um corpo e ainda, se a comparação dos rótulos desse corpo for bem sucedida (chamada recursiva à função de comparação) então os rótulos são estruturalmente iguais. Se na mesma posição relativa um

elemento for simples e o outro não, ou se a avaliação dos corpos dos dois elementos obtiver um resultado *falso* em relação à comparação de seus itens, então os rótulos são estruturalmente diferentes.

(v) **<TABLE>**: a comparação de duas tabelas é mais complexa do que a comparação de outros rótulos, e será explicada em detalhes como subseção deste capítulo.

3.4.1 Comparação entre duas tabelas

A situação mais simples que pode ocorrer com duas tabelas é aquela em que ambas têm a mesma quantidade de colunas em todas as linhas, independentemente do número de linhas. A quantidade de linhas não é relevante porque, no momento da criação da base de dados, objetivo deste protótipo, cada linha de uma tabela gerará uma tupla. Portanto, se duas tabelas possuem a mesma quantidade de colunas em todas as suas linhas, então elas são consideradas estruturalmente iguais, ou seja, possuem o mesmo esquema de banco de dados.

Entretanto, tabelas em documentos HTML podem apresentar e, na maioria dos documentos analisados na elaboração deste trabalho apresentaram realmente, configurações bastante diferentes daquela descrita como a mais simples. É bastante comum a situação de grupos de linhas de uma tabela conterem quantidade de colunas diferente das linhas anteriores ou posteriores. A questão é que, como afirmado anteriormente, a quantidade de linhas é irrelevante para a comparação entre duas tabelas. Isso significa que essa comparação deve ser feita de maneira mais abrangente do que simplesmente percorrendo seqüencialmente as linhas de duas tabelas e verificando se elas possuem a mesma quantidade de colunas nas mesmas

posições. Se a comparação fosse feita dessa maneira, as duas tabelas da **Figura 17** seriam consideradas diferentes.

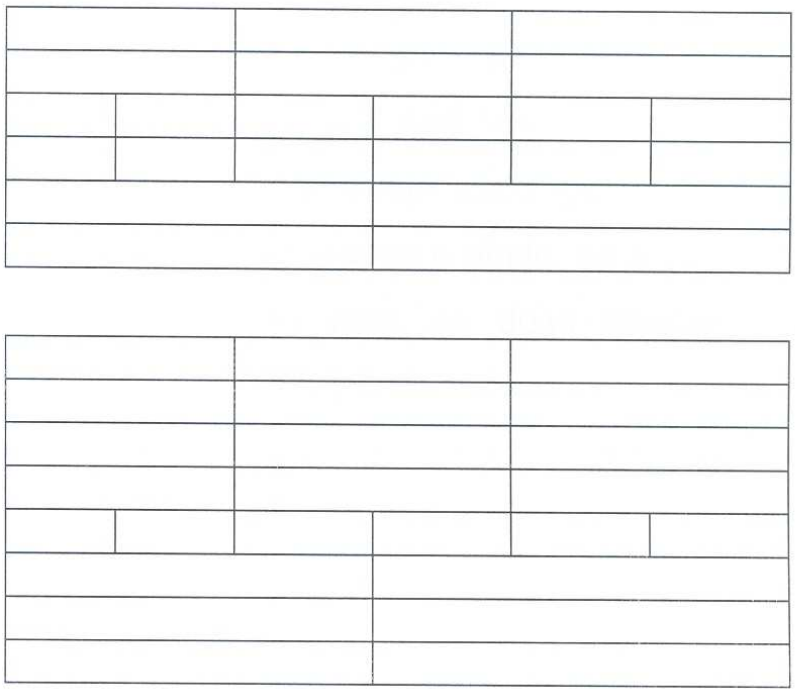


FIGURA 17 – TABELAS COM QUANTIDADE DIFERENTE DE COLUNAS EM CADA LINHA

Entretanto, as tabelas esquematizadas na **Figura 17** possuem a mesma estrutura interna, o que será demonstrado a seguir.

Denominando a primeira tabela da **Figura 17** de *Tabela1* e a segunda tabela da **Figura 17** de *Tabela2,* se for criada uma lista com a quantidade de colunas em cada linha de *Tabela1*, obtém-se o seguinte conjunto: {3, 3, 6, 6, 2, 2}. Fazendo o mesmo com *Tabela2* obtém-se: {3, 3, 3, 6, 2, 2, 2}.

Considerando cada conjunto de linhas com a mesma quantidade de colunas como um *grupo*, pode-se dizer que *Tabela1* possui três grupos, cuja quantidade de colunas em cada um é dada pelo seguinte conjunto: {3, 6, 2}. Submetendo *Tabela2* ao mesmo procedimento tem-se o conjunto: {3, 6, 2}. Isso prova que, estruturalmente, as duas tabelas são iguais, já que estão divididas na mesma quantidade de grupos, os quais possuem internamente a mesma quantidade de colunas. É como se as duas tabelas fossem, na

realidade, agrupamentos de tabelas simples, relembrando que, na comparação de tabelas simples, a quantidade de linhas de cada uma é irrelevante.

O protótipo faz esse tipo de análise nas tabelas para estabelecer a comparação entre as estruturas. Ele cria uma lista dos grupos existentes, identificando quantas colunas possuem cada grupo. Se as duas tabelas tiverem a mesma quantidade de grupos e ainda, se a quantidade de colunas de cada grupo for a mesma para as duas tabelas, então elas são consideradas iguais estruturalmente.

Entretanto, a comparação de tabelas não termina nesse ponto. A linguagem HTML permite que o conteúdo de uma célula seja composto de outros rótulos, inclusive de outras tabelas. Isso deve também ser levado em consideração no momento da comparação. Por exemplo, as tabelas da **Figura 18** possuem os mesmos agrupamentos, mas são estruturalmente diferentes.

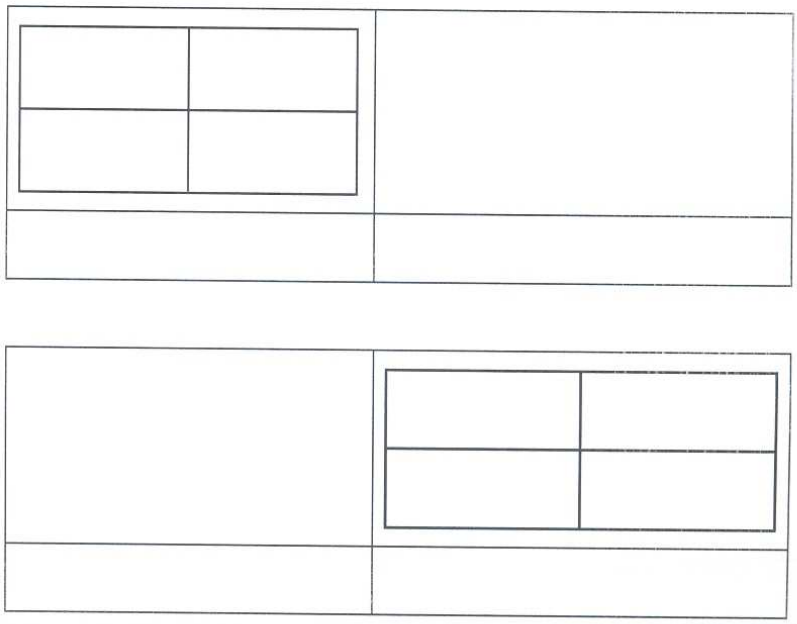


FIGURA 18 – TABELAS QUE POSSUEM CÉLULAS COM ESTRUTURA INTERNA

Embora as duas tabelas possuam a mesma configuração de agrupamentos (um único grupo com duas colunas), existem células que possuem estrutura interna cuja localização relativa no grupo é diferente.

Denominando a primeira tabela de *Tabela1* e a segunda de *Tabela2,* se for criada para *Tabela1* uma matriz contendo: na primeira coluna a posição relativa do grupo na tabela, na segunda coluna a posição relativa da linha na tabela e, na terceira coluna, a posição relativa da coluna na linha, somente para as células que possuem rótulos internos, obtém-se o seguinte:

Grupo	Linha	Coluna
0	0	**0**

Submetendo *Tabela2* ao mesmo processamento, obtém-se o seguinte:

Grupo	Linha	Coluna
0	0	**1**

As células que possuem corpo interno pertencem ao mesmo grupo, mas estão em colunas diferentes. Outra situação possível seria elas estarem na mesma coluna, mas em grupos diferentes. Em ambas as situações as tabelas não são estruturalmente iguais.

É importante observar que, mesmo que as células que possuem corpo estivessem em linhas diferentes, se elas pertencessem ao mesmo grupo e estivessem na mesma coluna, as tabelas seriam consideradas estruturalmente iguais.

É essa a análise que é feita pelo protótipo. As matrizes de configuração das células com corpos internos são criadas para as duas tabelas. Caso possuam quantidade de linhas diferente, as tabelas não são estruturalmente iguais, já que uma possui mais células com corpos internos do que a outra.

Se as duas matrizes tiverem a mesma quantidade de linhas, então verifica-se se a posição relativa ao grupo e à coluna é a mesma para todas as células. Em caso negativo, as tabelas são consideradas estruturalmente diferentes.

Em caso afirmativo, se a comparação dos rótulos desses corpos for bem sucedida (chamada recursiva à função de comparação) então as tabelas são estruturalmente iguais.

3.5 CRIAÇÃO DA BASE DE DADOS

Caso as duas versões do documento HTML sejam equivalentes estruturalmente, dois *scripts* de banco de dados serão criados: uma para os comandos DDL e outro para os comandos DML necessários à criação das relações e suas respectivas inclusões de dados.

O processamento realizado para a criação da base de dados é o seguinte:

(i) dois arquivos texto são criados. Os dois possuem como nome, o próprio nome do arquivo de uma das versões do documento (aquela identificada no protótipo como **Versão 1**) e, como extensão, .DDL e .DML. Cada um deles conterá, respectivamente, os comandos para criação das relações (ou o esquema do banco de dados) e os comandos para inclusão das informações nas relações que tiverem sido criadas;

(ii) depois o vetor de rótulos das duas versões será percorrido. Para cada dupla de rótulos serão executados os seguintes passos: execução do método *criaTabela* de um dos objetos e execução do método *insereDados* dos dois objetos. Isso será feito até que todos os rótulos tenham sido trabalhados.

Como aconteceu com a estruturação dos documentos, a criação da base de dados e também a inclusão de dados será feita de maneira diferente para cada rótulo. Na seqüência serão discutidos os procedimentos para realização dessas atividades.

Entretanto, antes de abordar os processamentos em particular, serão apresentados em formato tabular os parâmetros das funções *criaTabela* e *insereDados*, os quais serão citados posteriormente, e também discutidos alguns procedimentos que são comuns a todos os rótulos.

TABELA 5 – PARÂMETROS DAS FUNÇÕES *CRIATABELA* E *INSEREDADOS*

Parâmetro	Descrição
i	Posição relativa do item no vetor de rótulos do corpo sendo analisado.
arq	Exclusivo de *criaTabela*. Arquivo no qual devem ser gravados os comandos DDL.
chaveEstrangeira	Valor lógico que indica se a relação a ser criada ou aquela na qual estão sendo incluídos os dados possui relacionamento 1:N com a relação hierarquicamente superior.
hierarquia	Tem duas funções: ▪ Criar o nome da relação para a qual serão gerados os comandos CREATE TABLE ou INSERT INTO. ▪ Servir de parâmetro para a determinação do valor da chave estrangeira.
dt e hr	Exclusivo de *insereDados*. Representa a data e a hora da versão do documento. Necessários para atribuir o caráter temporal ao banco de dados.
ddl e dml	Exclusivo de *insereDados*. Representa os arquivos nos quais deverão ser gravados os comandos DDL e DML do banco de dados.

No momento de criação de uma relação é necessário determinar-lhe um nome. Neste protótipo, de maneira geral, uma tabela é denominada da seguinte forma: ***hierarquia + '_' + texto que descreve o*** rótulo ***+ i***, onde:

(i) *hierarquia* é o valor do parâmetro descrito na tabela anterior;

(ii) *texto que descreve o* rótulo é determinado por uma função interna do programa, a qual converte para texto o atributo **codigoTag** que todo rótulo possui;

(iii) *i* é a posição relativa do rótulo no vetor de rótulos do corpo que está sendo avaliado.

Por exemplo, a relação **Documento_Divisao9_Citacao1**[5] representa uma citação, a qual ocupa a segunda posição no corpo de uma divisão, a

[5] A implementação real utiliza nomes mais abreviados para as relações (por exemplo, Doc_Dv9_Ct1) por causa de limites impostos pelo SGBD utilizado na criação da base de dados. Nessa parte do texto, entretanto, serão utilizados nomes mais extensos a fim de facilitar a compreensão.

qual, por sua vez, ocupa a décima posição no corpo de um documento. As posições dos rótulos nos respectivos vetores inicia sempre em zero.

Outra possibilidade para o nome de uma relação seria aquela apresentada a seguir: **Documento_Paragrafo1_Corpo**. Nessa relação estarão as tuplas relacionadas ao parágrafo e que constituem seu corpo. Esse nome faz supor que existe uma relação chamada **Documento_Paragrafo1**, a qual contém os atributos do parágrafo que é hierarquicamente superior ao seu corpo, o que é verdade.

O motivo para esse fato é que, na abordagem relacional para bancos de dados implementada nos SGBDs atuais, cada valor na relação, isto é, cada valor de atributo em cada tupla é *atômico*, o que significa dizer que, dada uma posição linha/coluna de uma relação, somente poderá haver um único valor[6]. Sabe-se que um parágrafo pode ter um corpo. Lembrando o fato de que um corpo é, na realidade, um conjunto de rótulos, entre eles até mesmo uma outra tabela, não seria possível a criação de uma relação que contivesse todo o corpo de um rótulo, já que o valor de um atributo tem de ser atômico.

A solução proposta é, de maneira geral, a de criar uma relação para o corpo de um rótulo ligada à primeira por meio de uma chave estrangeira, uma vez que o relacionamento que se estabelece entre as duas relações será sempre de cardinalidade 1:N.

Por exemplo, voltando à situação do parágrafo que possui um corpo, a **Figura 19** representa o Diagrama Entidade-Relacionamento (DER[7]) que poderia ser criado.

---

[6] Para DATE em [10] e [11], o valor de um atributo deve pertencer a um domínio, o qual representa um **tipo de dado**. Tal tipo de dado pode ser definido pelo sistema ou pelo usuário, o que permitiria a criação de tipos bastante complexos. Entretanto, a tecnologia implementada atualmente nos SGBDs relacionais ainda não incorporou esses pressupostos. Portanto, fica assumido que o valor de um atributo tem de ser *atômico* no sentido em que esse termo ainda é compreendido nas implementações de bancos de dados relacionais existentes.

[7] A simbologia utilizada para a criação dos Diagramas Entidade-Relacionamento foi retirada de [5].

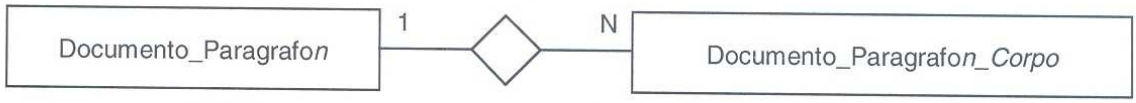


FIGURA 19 – DER QUE REPRESENTA PARÁGRAFO COM CORPO

É esse o objetivo do parâmetro *chaveEstrangeira*, nos métodos *criaTabela* e *insereDados*. Ele causa o seguinte efeito nos dois métodos, caso tenha conteúdo verdadeiro:

(i) Na geração do comando CREATE TABLE ele simplesmente ocasiona a inclusão do campo **ChaveEstrangeira** do tipo INTEGER na relação.

(ii) Na geração do(s) comando(s) INSERT INTO, ele faz com que um comando seja incluído e outro alterado no *script* de banco de dados:

a) **SELECT (Max) Chave AS UltimaChave FROM *hierarquia*;**, onde *hierarquia* representa o valor do parâmetro de mesmo nome, ou seja, o nome da relação com a qual aquela que está sendo atualizada possui um relacionamento 1:N. Esse comando será incluído e tem o objetivo de resgatar o valor efetivo da chave estrangeira a ser incluído na relação a fim de implementar o relacionamento. Como a seqüência de inclusão das tuplas é conhecida, sabe-se que o último registro é aquele que efetivamente deve ser tomado como valor da chave estrangeira.

b) **PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira INTEGER;**. O comando PARAMETERS será sempre criado, já que é a maneira de atribuir valor à chave primária da relação na qual se deseja incluir uma tupla. Caso o parâmetro *chaveEstrangeira* possua valor lógico verdadeiro, ele será alterado para incluir também essa informação.

Por fim, o último item que deve ser comentado antes da explicação dos mecanismos particulares utilizados na criação dos *scripts* de banco de dados é a determinação do valor a ser incluído como chave primária de uma relação.

Sabe-se que o valor de uma chave primária não pode ser nulo. Portanto, no momento da inclusão de uma tupla no banco de dados é necessário informar um valor para ser atribuído àquele atributo. Entretanto, o campo *ChavePrimaria* de uma tabela é do tipo INTEGER e deve ser determinado em função dos outros registros já incluídos na tabela. Isso significa dizer que ele, obviamente, não fará parte do código HTML do qual foi extraída a informação.

Portanto, o protótipo deveria ter uma maneira de retirar do próprio banco de dados já criado, o valor relativo à chave primária da nova tupla como sendo uma unidade a mais do que a chave primária da última tupla incluída. Pensou-se na possibilidade de utilizar um tipo de dado auto-numerado, o que eliminaria esse processo. Entretanto, o valor da chave estrangeira também deve ser buscado na tabela que representa o *lado 1* do relacionamento, dessa forma, não poderia ser um campo auto-numerado.

Diante desse fato, optou-se pela implementação SQL da Microsoft, presente nos produtos Visual Basic e Access. Nessa implementação, é possível criar o que é chamado de *consulta parâmetro* [1][23] e isso é feito pelo comando PARAMETERS.

O comando PARAMETERS possibilita a criação das consultas parâmetro, por meio das quais é possível criar comandos SQL que possuam nomes de parâmetros no lugar dos valores efetivos que devem ser incluídos no banco de dados. O valor desse parâmetro deverá ser determinado antes que a consulta propriamente dita seja executada e isso, no protótipo, é feito por um comando SELECT. Por exemplo, para incluir uma tupla na tabela que

representa o parágrafo discutido anteriormente, a seguinte seqüência de comandos é gravada no *script* de comandos DML:

```
SELECT MAX(Chave)AS UltimaChave
FROM Documento_Paragrafo1;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Documento_Paragrafo1 (Chave, Data, Hora)
VALUES (ChavePrimaria, '21/4/2001', '10:30:32');
```

A ferramenta que executa os *scripts* de banco de dados, criada em Visual Basic, em primeiro lugar executa a consulta seleção [1][23] que devolve o maior valor existente na chave primária da tabela. No resultado, esse campo aparecerá com o nome *UltimaChave*. Ao conteúdo desse atributo será acrescentada uma unidade (caso seja o primeiro registro a ser incluído na tabela, a chave primária terá valor 1). Esse valor é atribuído ao parâmetro *ChavePrimaria* e a consulta parâmetro representada pelos comandos PARAMETERS e INSERT INTO é executada.

No caso de uma tabela que possua chave estrangeira, como aquela do corpo discutida anteriormente, o seguinte conjunto de comandos é gerado no *script* de banco de dados:

```
SELECT MAX(Chave) AS UltimaChave FROM
Documento_Paragrafo1_Corpo;

SELECT MAX(Chave) AS UltimaChave FROM
Documento_Paragrafo1;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Documento_Paragrafo1_Corpo (Chave, Texto,
Data, Hora, ChaveEstrangeira) VALUES (ChavePrimaria,
'Uma citação. ', '21/4/2001', '10:30:32',
ChaveEstrangeira);
```

3.5.1 Criação de comandos DDL e DML para parágrafos

Na análise de um parágrafo três situações podem ocorrer. Em cada uma delas será tomada uma atitude diferente quanto à criação da tabela e inclusão dos dados. São elas:

(i) **O parágrafo é simples**, ou seja, ele não contém um corpo. Nesse caso será criado um comando CREATE TABLE e um comando INSERT INTO.

(ii) **O parágrafo contém um corpo constituído somente de parágrafos simples**. Nesse caso, será criado somente um comando CREATE TABLE e tantos comandos INSERT INTO quantos forem os rótulos internos do parágrafo.

(iii) **O parágrafo contém um corpo constituído de listas, tabelas ou de elementos que possuem seu próprio corpo**. Será criado um comando CREATE TABLE para o parágrafo contendo somente sua identificação (chave primária, data e hora). Para cada rótulo que representar lista, tabela ou que possuir um corpo, será criada uma tabela, relacionada à primeira por meio de chave estrangeira. Todos os rótulos que forem simples ficarão reunidos numa tabela cujo nome será o mesmo da tabela original do parágrafo acrescido do sufixo *_Corpo*. Os comandos INSERT INTO são criados de acordo com essa estrutura. O seguinte DER mostra como ficariam as tabelas de banco de dados para uma divisão que possuísse internamente uma tabela, uma lista simples e dois parágrafos simples, nesta ordem:

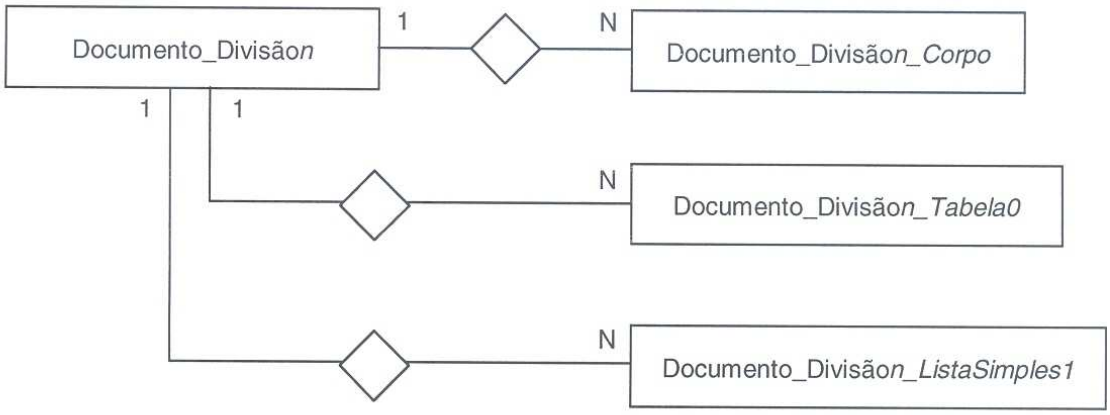


FIGURA 20 – DER QUE REPRESENTA DIVISÃO COM CORPO

A criação das tabelas para a lista simples e para a própria tabela é feita pelos respectivos métodos dos objetos.

3.5.2 Criação de comandos DDL e DML para listas simples

Uma lista simples não pode conter um corpo, portanto, como o próprio nome diz, sua criação em base de dados é simples.

Um comando CREATE TABLE é gravado. A tabela possuirá os seguinte atributos: *Chave, Linha, Data e Hora*. Caso a lista faça parte do corpo de outro rótulo, também haverá o atributo *ChaveEstrangeira*.

Depois serão gerados tantos comandos INSERT INTO quantas forem as linhas da lista.

3.5.3 Criação de comandos DDL e DML para listas de definição

A tabela para criação em base de dados de uma lista de definição é sempre simples. Possui os atributos *Chave, Definição, Descrição, Data* e *Hora*, além de *Chave Estrangeira*, se for parte integrante de um rótulo que possua corpo.

É na inclusão dos dados nessa tabela que ocorre um processamento diferenciado. Será incluída exatamente uma tupla para cada linha da lista de definição. Entretanto, embora uma definição seja sempre simples, uma

descrição pode conter, ela mesma, um corpo. Poderia ter sido implementado o mesmo raciocínio usado para as chaves estrangeiras. Entretanto, não é todo o item da lista que possui um relacionamento com outra tabela, mas somente a descrição de um item. O corpo é o conteúdo do atributo descrição e não um relacionamento de todo o item com outros rótulos.

Por esse motivo, a solução proposta é a de, para cada rótulo que faça parte do corpo de uma descrição, criar uma tabela e, no atributo descrição daquele item, incluir a mensagem *"Possui corpo. Tabelas começam com* <Nome da tabela da lista de definição>".

3.5.4 Criação de comandos DDL e DML para textos pré-formatados

Textos pré-formatados não podem ter corpo interno. Portanto, sua criação em base de dados é simples como a criação em base de dados de uma lista simples. A tabela conterá os atributos: *Chave, Texto, Data* e *Hora*, além de *ChaveEstrangeira* se o rótulo fizer parte do corpo de outro rótulo.

Depois um único comando INSERT INTO será gerado a fim de incluir o texto no banco de dados.

3.5.5 Criação de comandos DDL e DML para tabelas

Há quatro situações possíveis para uma tabela:

(i) Ela possui um único grupo e nenhuma célula desse grupo possui corpo interno.

(ii) Ele possui mais de um grupo, mas nenhuma célula desses grupos possuem corpos internos.

(iii) Ela possui um único grupo, mas há pelo menos uma célula que possui corpo interno.

(iv) Ela possui vários grupos e há pelo menos uma célula que possui corpos internos.

Na situação *(i)*, será gerado apenas um comando CREATE TABLE e tantos comandos INSERT INTO quantas forem as linhas da tabela.

A situação *(ii)* caracteriza um agrupamento de tabelas, portanto, cada grupo dará origem a um comando CREATE TABLE. Em cada grupo haverá tantos comandos INSERT INTO quantas forem as linhas do grupo. Além dessas tabelas, existirá uma outra, a qual pode-se chamar *tabela mestre*, que agrupa todas as outras por meio de chave estrangeira. Por exemplo, se houver uma tabela com três grupos internos, o seguinte DER demonstra como ficará o banco de dados:

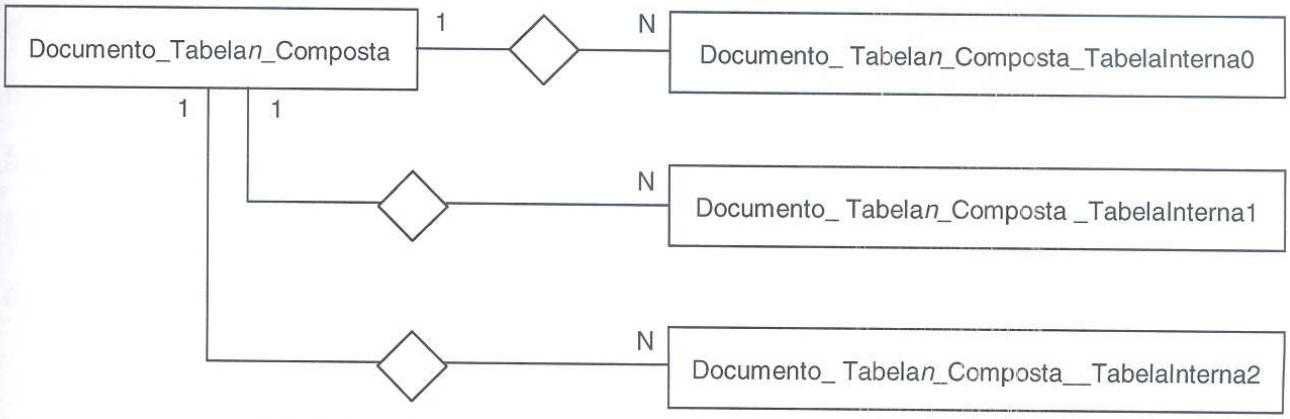


FIGURA 21 – DER QUE MOSTRA TABELA COMPOSTA DE GRUPOS

A situação *(iii)* gera um processamento parecido com o da situação *(i)*. A diferença está nos comandos INSERT INTO. A coluna que possui uma célula com corpo será obrigatoriamente do tipo LONGTEXT. O seu conteúdo será composto de um texto padrão do tipo: "*Possui corpo. Tabelas começam com* <nome da tabela atual>".

A situação *(iv)* gera um processamento semelhante ao da situação *(ii)*, acrescentando o fato descrito na situação *(iii)*: as colunas que possuírem corpo serão obrigatoriamente do tipo LONGTEXT e o conteúdo da célula será composto do mesmo texto padrão.

Ainda para o processamento de tabelas e, independentemente da situação à qual ela pertença, é necessário discutir o modo pelo qual são decididos o nome da coluna e seu tipo de dados.

Para todos os outros rótulos, os nomes de coluna são padronizados, bem como o tipo de dados: INTEGER para as chaves primária e estrangeira (quando houver), LONGTEXT para os textos e DATETIME para data e hora, característica temporal do banco de dados.

Entretanto, para as tabelas não seria interessante adotar o mesmo procedimento padrão para todas as colunas. É possível que todas as informações de uma coluna sejam numéricas, ou representem uma data. Criar sempre as colunas com tipo de dado LONGTEXT faria com que o banco de dados gerado tivesse necessariamente de passar por um processamento manual de mudança de tipo de dados, caso o usuário do sistema desejasse fazer consultas que levassem em consideração tipos numéricos e datas. Muito provavelmente, as informações que mais interessarão ao futuro usuário de um sistema criado com esta tecnologia estarão nas tabelas.

O que esse protótipo faz é inferir o tipo de dado de uma coluna, com base nos valores que ela contém. As situações que podem acontecer e o procedimento adotado para cada uma delas estão descritos a seguir:

(i) Em todas as linhas as informações de uma coluna são textuais ou são inexistentes (valores nulos): nesse caso o tipo de dado da coluna é LONGTEXT e nome da coluna é genérico: *Coluna + posição relativa da coluna na linha.*

(ii) Em todas as linhas as informações são numéricas inteiras ou inexistentes (valores nulos): nesse caso o tipo de dado da coluna é INTEGER e o nome da coluna é genérico: *Coluna + posição relativa da coluna na linha.*

(iii) Em todas as linhas as informações são numéricas reais, inteiras ou inexistentes (valores nulos): nesse caso o tipo de dado da coluna é DOUBLE e o nome da coluna é genérico: *Coluna + posição relativa da coluna na linha*.

(iv) Em todas as linhas as informações representam datas ou horas, ou ainda, são inexistentes (valores nulos): nesse caso o tipo de dado da coluna é DATETIME e o nome da coluna é genérico: *Coluna + posição relativa da coluna na linha*.

(v) Somente na primeira linha a informação é textual e, em todas as outras linhas, as informações pertencem a um mesmo domínio (INTEGER, DOUBLE ou DATETIME): nesse caso, o tipo da coluna será aquele determinado pelo domínio das outras linhas, exceto a primeira, e o nome da coluna será o conteúdo da primeira linha.

3.5.6 Exemplo de resultado de criação de esquema de base de dados

Com o objetivo de tornar claro qual é o *script* de banco de dados obtido pelo processamento sobre duas versões de um mesmo documento HTML, serão apresentadas duas possíveis versões de um mesmo documento e o *script* de banco de dados gerado com bases nessas versões. De forma análoga ao que foi feito no exemplo apresentado na subseção 3.3.6, os atributos dos rótulos foram eliminados, bem como rótulos que não são considerados no processamento. O motivo foi diminuir a complexidade da comparação entre o código de origem e o respectivo resultado.

A **Figura 22** mostra o início do código HTML da primeira versão documento (o mesmo apresentado em 3.3.6). O código completo pode ser consultado no **Anexo 2**.

```
<html>
<body>

<p>Este é um parágrafo de texto normal. O
próximo contém um corpo interno:</p>

<p>
   <cite>Uma citação.</cite>
   <code>Uma linha de código</code>
   <cite>Outra citação.</cite>
</p>

...
</body>
</html>
```

FIGURA 22 – TRECHO DE CÓDIGO HTML DA VERSÃO 1

A **Figura 23** mostra um trecho do código da segunda versão do mesmo documento. O código completo pode ser consultado no **Anexo 2**. Há algumas modificações em termos de conteúdo e também de quantidade de linhas em tabelas e listas.

```
<html>
<body>

<p>Este é um parágrafo de texto normal. O
próximo contém um corpo interno (versão
2):</p>

<p>
   <cite>Uma citação (v2).</cite>
   <code>Uma linha de código (v2)</code>
   <cite>Outra citação (v2).</cite>
</p>

...
</body>
</html>
```

FIGURA 23 – TRECHO DE CÓDIGO HTML DA VERSÃO 2

O processamento dessas duas versões de um mesmo documento

gerou o *script* de banco de dados apresentado na **Figura 24** e na **Figura 25**[8].

```
CREATE TABLE Doc_Pg0 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data
DATE, Hora TIME);

CREATE TABLE Doc_Pg1 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Data DATE, Hora TIME);

CREATE TABLE Doc_Pg1_Corpo (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data
DATE, Hora TIME, ChaveEstrangeira INTEGER CONSTRAINT
Doc_Pg1_Corpo REFERENCES Doc_Pg1);[9]

...

CREATE TABLE Doc_LD9_Dv2 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data
DATE, Hora TIME);
```

FIGURA 24 – CÓDIGO DDL

```
SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg0 (Chave, texto, data, hora) VALUES
(ChavePrimaria, 'Este é um parágrafo de texto normal.
O próximo contém um corpo interno: ', '21/8/2001',
'15:33:20');

...

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9_Dv2;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_LD9_Dv2 (Chave, texto, data, hora)
VALUES (ChavePrimaria, 'Uma divisão (v2) ',
'20/8/2001', '16:38:46');
```

FIGURA 25 – CÓDIGO DML

Na **Figura 26** são mostrados os relacionamentos estabelecidos entre algumas relações no banco de dados. Mais especificamente, aquelas que possuíam um corpo, como um dos parágrafos e a divisão e também a tabela composta de mais de um grupo. Outras relações também foram criadas no

[8] O código completo pode ser consultado no **Anexo 3**.

[9] A instrução CONSTRAINT *nome* REFERENCES *relaçãoexterna* estabelece um relacionamento 1:N entre a tabela que está sendo criada e aquela referenciada como *relaçãoexterna*. Isto impõe integridade referencial, entretanto, essa cláusula não estabelece alterações e inclusões em cascata, ou seja, uma tupla não poderá ser excluída de uma relação se tiver relacionamentos com outras relações [1].

banco de dados, mas não aparecem na figura por motivo de simplificação[10].

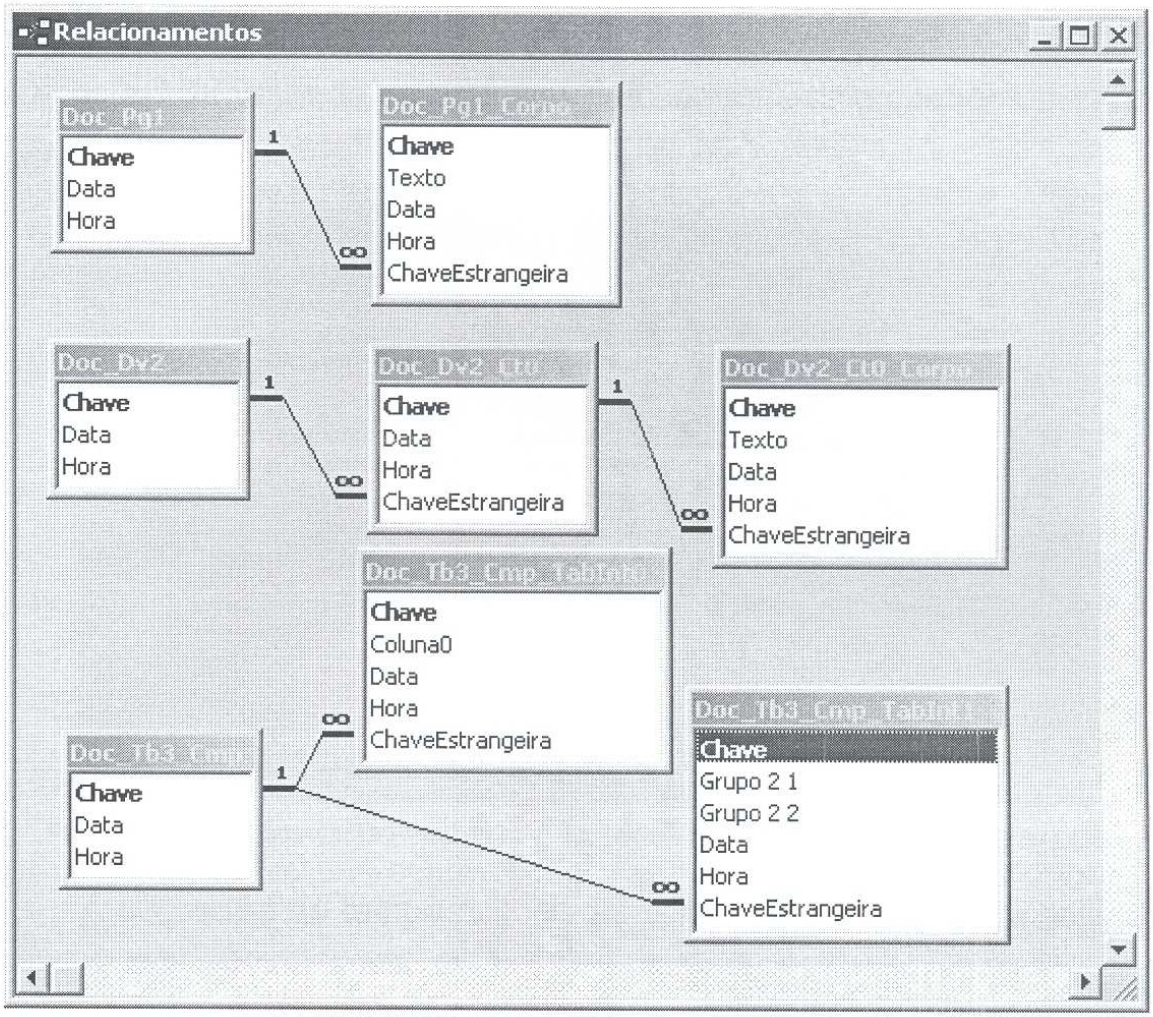


FIGURA 26 – RELACIONAMENTOS ENTRE AS RELAÇÕES CRIADAS PELO *SCRIPT* DE BANCO DE DADOS

Esta seção mostrou um exemplo composto por duas versões de um mesmo documento e como ficaria o *script* de criação do banco de dados e inclusão de tuplas na mesma base.

3.6 PREENCHIMENTO DA BASE DE DADOS JÁ CRIADA

O mesmo protótipo que é usado para criar a base de dados, o é também para o preenchimento posterior dessa base. A **Figura 27** mostra a interface do programa.

Na área marcada como **Primeiro processamento** há dois botões de

[10] O conteúdo completo de todas as relações está no **Anexo 4**.

opção. Ao escolher a opção **Sim**, o usuário indica que devem ser gerados tanto os comando DDL quando os comandos DML. Escolhendo a opção **Não** o usuário estará informando ao protótipo que somente os comandos DML devem ser gerados.

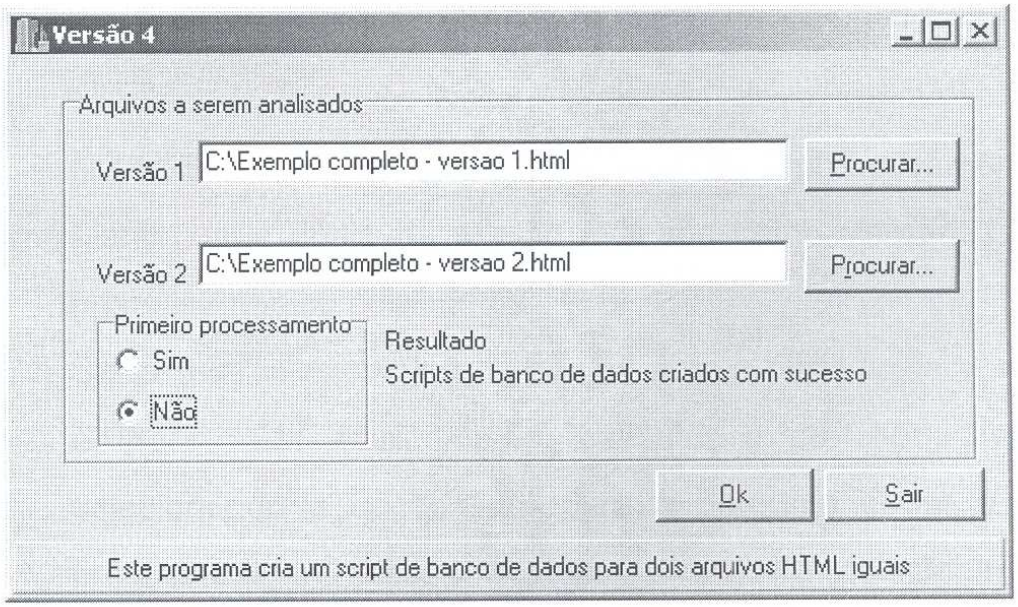


FIGURA 27 – INTERFACE DO PROTÓTIPO PARA GERAÇÃO E/OU INCLUSÃO DE DADOS

A dimensão de tempo associada às tabelas a fim de que seja criado um banco de dados do tipo tempo de transação com caráter temporal associado às tuplas, conforme estabelecido no capítulo 2, não aparece nessa tela porque é retirada diretamente das informações de data e hora de gravação do arquivo.

No momento da criação da estrutura de um documento suas informações de data e hora são extraídas e armazenadas nos atributos de mesmo nome da classe *ClDocumento*. Essas informações são passadas aos métodos de criação dos comandos INSERT INTO a fim de que sejam incluídas nos comandos que atualizarão a base de dados. Isso caracteriza a representação temporal explícita com variação temporal discreta ponto a ponto, conforme discutido no capítulo 2.

O procedimento a ser adotado para criação e preenchimento da base de dados é o seguinte:

(i) Salvar duas versões da página HTML para a qual deve ser feito o processamento.

(ii) Caso a página possua *frames*, identificar o arquivo no qual estão as informações de interesse.

(iii) Executar o protótipo indicando a localização no disco das duas versões do arquivo. Nessa execução deve ser selecionada a opção **Sim** para a pergunta sobre o primeiro processamento.

(iv) Manter uma cópia da versão indicada como **versão 1** para os processamentos posteriores.

(v) Salvar outros exemplos da página com a periodicidade desejada e, para cada nova versão, executar o protótipo, dessa vez selecionando a opção **Não** para a pergunta sobre o primeiro processamento. Nessa ocasião, o arquivo selecionado como **versão 1** deverá ser o mesmo utilizado na criação da base de dados.

A simples inclusão de novos dados no banco de dados gerado a partir de versões de documentos HTML segue os mesmos passos iniciais da criação do banco de dados como um todo: estruturação dos dois documentos e comparação das estruturas. A diferença está na atualização do banco de dados: somente os comandos para inclusão de novos registros serão gerados, e somente para a segunda versão, já que a primeira foi processada anteriormente.

É necessário manter a *versão 1* para que se possa saber se o documento sofreu mudanças estruturais. O protótipo, implementado em C++ Builder, não tem acesso ao banco de dados gerado, ele simplesmente cria um *script* de comandos SQL, os quais são executados por um outro aplicativo, parte do protótipo, criado em Visual Basic. Essa foi a maneira encontrada para comparação da nova versão do documento com sua versão original.

A ferramenta que executa os *scripts* de banco de dados tem interface semelhante à do protótipo de reconhecimento da estrutura e geração do *script* como mostrado na **Figura 28**.

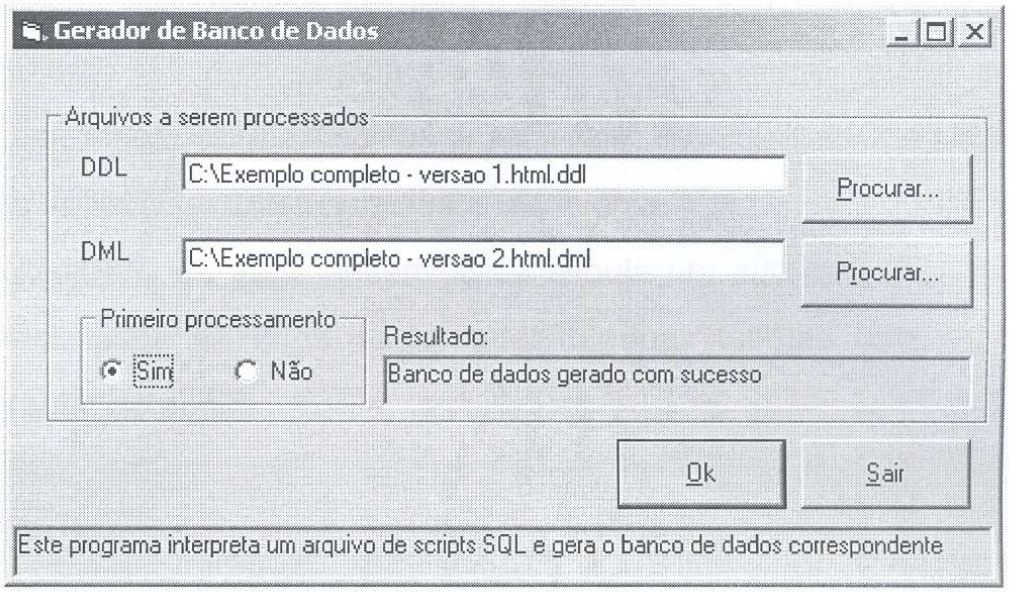


FIGURA 28 – INTERFACE DA FERRAMENTA DE EXECUÇÃO DOS *SCRIPTS*

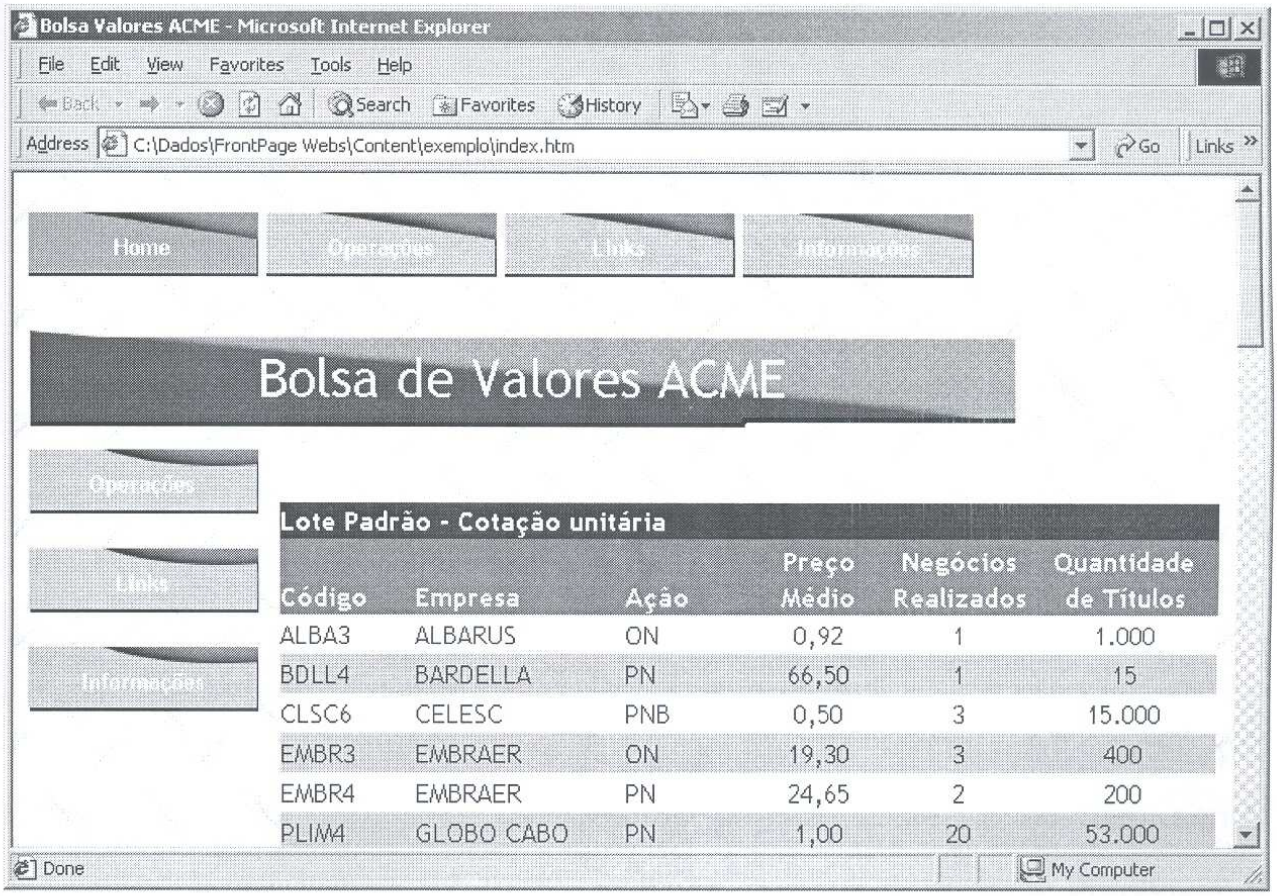
A exemplo do que aconteceu na primeira interface apresentada, aqui também há a pergunta se é o primeiro processamento, ou não. Se for, o usuário terá de escolher o arquivo com a DDL e aquele com a DML. Caso não seja o primeiro processamento, a caixa de textos que permite a indicação do arquivo com a DDL será desabilitada e o usuário somente poderá escolher o arquivo que contém os comandos DML.

Este capítulo apresentou o protótipo desenvolvido como parte deste trabalho. Mostrou-se como é feito o reconhecimento da estrutura de um documento HTML a partir de um subconjunto dos rótulos desta linguagem. Depois, indicou-se como essa estrutura é traduzida em comandos SQL, os quais são armazenados em um arquivo texto que é depois processado para a criação efetiva da base de dados.

4 RESULTADOS OBTIDOS

Neste capítulo serão demonstrados os resultados obtidos como processamento de uma página de teste, especialmente criada para este trabalho, e ainda com duas páginas reais da Internet. Também serão discutidas aplicações potenciais desta tecnologia.

A página de teste desenvolvida simula um *site* de bolsa de valores. Ela contém, além de itens de formatação, duas tabelas com informações sobre cotação de ações.



Lote Padrão - Cotação unitária

Código	Empresa	Ação	Preço Médio	Negócios Realizados	Quantidade de Títulos
ALBA3	ALBARUS	ON	0,92	1	1.000
BDLL4	BARDELLA	PN	66,50	1	15
CLSC6	CELESC	PNB	0,50	3	15.000
EMBR3	EMBRAER	ON	19,30	3	400
EMBR4	EMBRAER	PN	24,65	2	200
PLIM4	GLOBO CABO	PN	1,00	20	53.000

FIGURA 29 – PÁGINA HTML USADA DE EXEMPLO

O processamento desta página feito pelo protótipo desenvolvido gerou as relações de banco de dados descritas na **Tabela 6**.

TABELA 6 – TABELAS GERADAS COMO RESULTADO DO PROCESSAMENTO DA PÁGINA DE EXEMPLO

Nome da Tabela	Atributo	Significado do Atributo
Doc_Pg0	Chave	Chave primária
	Texto	Conteúdo do parágrafo
	Data	Data de gravação do documento
	Hora	Hora de gravação do documento
Doc_Pg2	Chave	Chave primária
	Texto	Conteúdo do parágrafo
	Data	Data de gravação do documento
	Hora	Hora de gravação do documento
Doc_Tb1	Chave	Chave primária
	Coluna0	Conteúdo da primeira coluna da tabela
	Coluna1	Conteúdo da segunda coluna da tabela
	Coluna2	Conteúdo da terceira coluna da tabela
	Coluna3	Conteúdo da quarta coluna da tabela
	Negócios Realizados	Conteúdo da quinta coluna da tabela
	Quantidade de Títulos	Conteúdo da sexta coluna da tabela
	Data	Data de gravação do documento
	Hora	Hora de gravação do documento
Doc_Tb3	Chave	Chave primária
	Coluna0	Conteúdo da primeira coluna da tabela
	Coluna1	Conteúdo da segunda coluna da tabela
	Coluna2	Conteúdo da terceira coluna da tabela
	Coluna3	Conteúdo da quarta coluna da tabela
	Negócios Realizados	Conteúdo da quinta coluna da tabela
	Quantidade de Títulos	Conteúdo da sexta coluna da tabela
	Data	Data de gravação do documento
	Hora	Hora de gravação do documento

Caso a página contivesse elementos de formatação, como, por exemplo, tabelas com links para outras páginas, estes acabariam sendo incorporados ao banco de dados resultante, o que poderia prejudicar a compreensão por parte de um potencial usuário deste sistema. Esse fato é observado quando o protótipo é executado com base em páginas reais da Internet.

Por exemplo, o endereço www.bloomberg.com/markets/wei.html contém informações sobre o mercado mundial de ações. A aparência parcial

da página é demonstrada na **Figura 30**.

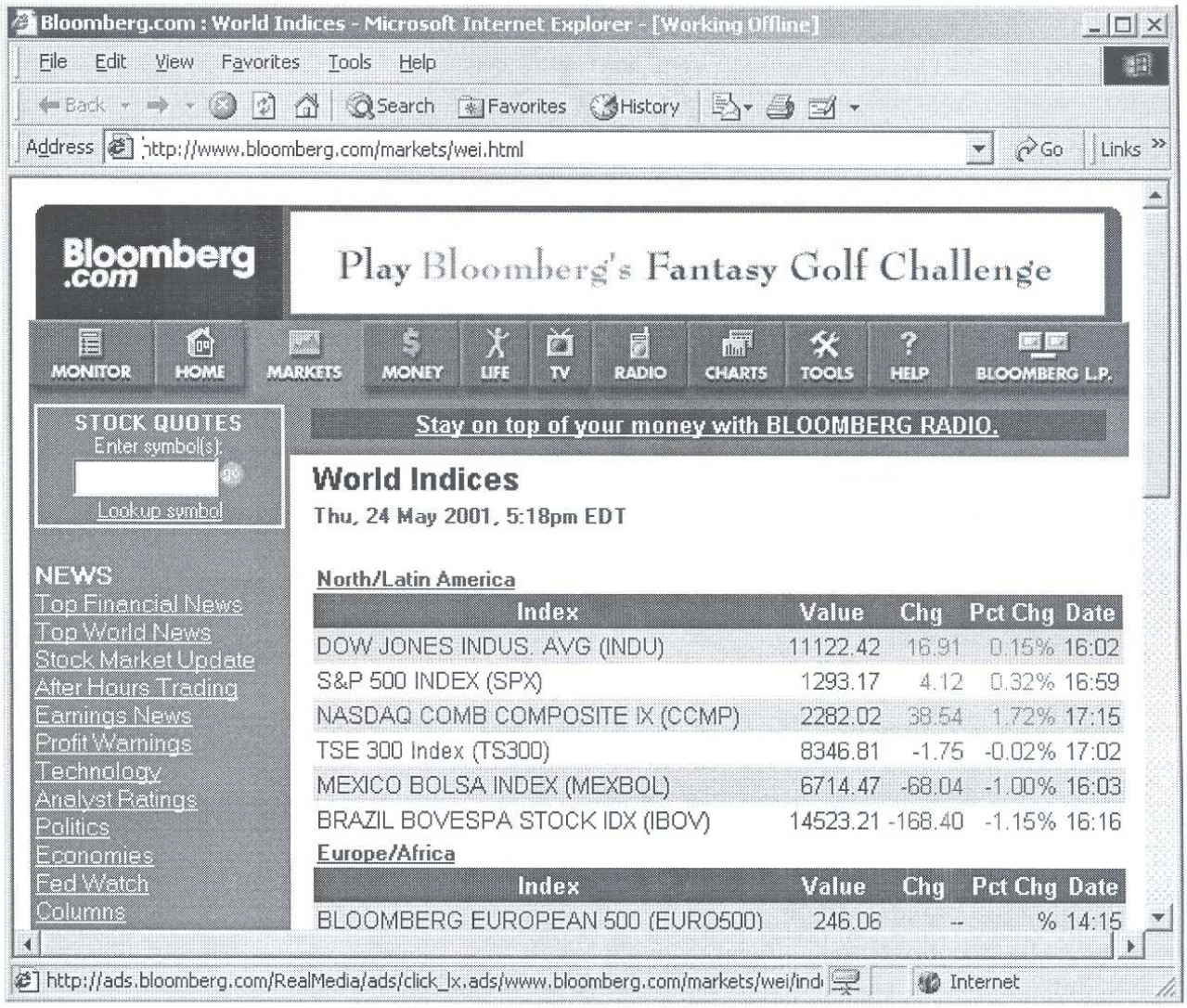


FIGURA 30 – PÁGINA DE EXEMPLO: COTAÇÃO DE AÇÕES

Embora invisível para o usuário, a formatação desta página é feita totalmente com base em tabelas aninhadas. O resultado do processamento desta página pelo protótipo deu origem ao seguinte banco de dados descrito na **Tabela 7**.

TABELA 7 – RESULTADO DO PROCESSAMENTO DA PÁGINA DA BLOOMBERG

Nome da Tabela	Atributos	Significado do nome da tabela
Doc_Tb0_Cmp	Chave, Data, Hora	Primeiro rótulo do documento é uma tabela que possui grupos internos.
Doc_Tb0_Cmp_TabInt0	Chave, Coluna0, Coluna1, Coluna2, Coluna3, Data, Hora, ChaveEstrangeira	Primeiro grupo da tabela composta Doc_Tb0_Cmp.
Doc_Tb0_Cmp_TabInt1	Chave, Coluna0, Data, Hora, ChaveEstrangeira	Segundo grupo da tabela composta Doc_Tb0_Cmp.
Doc_Pg2	Chave, Texto, Data, Hora	Segundo rótulo do documento é um parágrafo.

Nome da Tabela	Atributos	Significado do nome da tabela
Doc_Tb1	Chave, Coluna0, Coluna1, Data, Hora	Terceiro rótulo do documento é uma tabela.
Doc_Tb1_Ln0_Cl0_Pg1	Chave, Texto, Data, Hora	A relação Doc_Tb1 possui no endereço 0,0 (linha zero, coluna zero) um corpo. Esta é a primeira relação relativa a este corpo. Seu rótulo de origem é um parágrafo.
Doc_Tb1_Ln0_Cl0_Pg2	Chave, Texto, Data, Hora	Idem à relação anterior. Segundo rótulo do corpo daquela célula da tabela.
Doc_Tb1_Ln0_Cl0_Pg3	Chave, Texto, Data, Hora	Idem à relação anterior. Terceiro rótulo do corpo daquela célula da tabela
Doc_Tb1_Ln0_Cl0_Pg4	Chave, Texto, Data, Hora	Idem à relação anterior. Quarto rótulo do corpo daquela célula da tabela.
Doc_Tb1_Ln0_Cl0_Tb0	Chave, Coluna0, Data, Hora	Idem à relação anterior, com a diferença de que o rótulo de origem é uma tabela. Quinto rótulo do corpo daquela tabela.
Doc_Tb1_Ln0_Cl0_Tb0_Ln0_Cl0_Tb0	Chave, Coluna0, Data, Hora	A relação Doc_Tb1_Ln0_Cl0_Tb0 possui no endereço 0,0 (linha zero, coluna zero) um corpo constituído de uma tabela. Esta é a entidade do banco de dados responsável por representá-la.
Doc_Tb1_Ln0_Cl1_Tb0	Chave, Coluna0, Data, Hora	A relação Doc_Tb1 possui no endereço 0,1 (linha 0, coluna 1) um corpo constituído de uma tabela. Esta é a entidade do banco de dados responsável por representá-la.
Doc_Tb1_Ln1_Cl1_Tb0	Chave, Coluna0, Coluna1, Coluna2, Data, Hora	A relação Doc_Tb1 possui no endereço 1,1 (linha 1, coluna 1) um corpo. Esta é a primeira relação relativa a este corpo. Seu rótulo de origem é um parágrafo.
Doc_Tb1_Ln1_Cl1_Tb1_Cmp	Chave, Data, Hora	Idem à relação anterior. Segundo rótulo do corpo daquela célula da tabela. É constituído por uma tabela composta de vários grupos.
Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt0	Chave, Coluna0, Data, Hora, ChaveEstrangeira	Primeiro grupo de Doc_Tb1_Ln1_Cl1_Tb1.
Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt1	Chave, Coluna0, Value, Chg, Coluna3, Date, Data, Hora, ChaveEstrangeira	Segundo grupo de Doc_Tb1_Ln1_Cl1_Tb1.
Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt2	Chave, Coluna0, Data, Hora, ChaveEstrangeira	Terceiro grupo de Doc_Tb1_Ln1_Cl1_Tb1
Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt3	Chave, Coluna0, Value, Chg, Coluna3, Date, Data, Hora, ChaveEstrangeira	Quarto grupo de Doc_Tb1_Ln1_Cl1_Tb1.
Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt4	Chave, Coluna0, Data, Hora, ChaveEstrangeira	Quinto grupo de Doc_Tb1_Ln1_Cl1_Tb1

Nome da Tabela	Atributos	Significado do nome da tabela
Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt5	Chave, Coluna0, Value, Chg, Coluna3, Date, Data, Hora, ChaveEstrangeira	Sexto grupo de Doc_Tb1_Ln1_Cl1_Tb1

As informações de interesse estão localizadas nas seguintes relações:

- Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt1.
- Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt3.
- Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt5.

As outras relações contêm, em sua maioria, campos nulos. Isso acontece porque a formatação da página é feita com base em tabelas cujas células possuem imagens. As próprias tabelas de ações possuem uma formatação escondida apresentada na **Figura 31**.

World Indices
Thu, 24 May 2001, 5:18pm EDT

North/Latin America				
Index	**Value**	**Chg**	**Pct Chg**	**Date**
DOW JONES INDUS. AVG (INDU)	11122.42	16.91	0.15%	16:02
S&P 500 INDEX (SPX)	1293.17	4.12	0.32%	16:59
NASDAQ COMB COMPOSITE IX (CCMP)	2282.02	38.54	1.72%	17:15
TSE 300 Index (TS300)	8346.81	-1.75	-0.02%	17:02
MEXICO BOLSA INDEX (MEXBOL)	6714.47	-68.04	-1.00%	16:03
BRAZIL BOVESPA STOCK IDX (IBOV)	14523.21	-168.40	-1.15%	16:16
Europe/Africa				
Index	**Value**	**Chg**	**Pct Chg**	**Date**
BLOOMBERG EUROPEAN 500 (EURO500)	246.06	--	%	14:15
FT-SE 100 Index (UKX)	5915.90	18.50	0.31%	11:36
CAC 40 INDEX (CAC)	5656.47	25.73	0.46%	12:28
DAX INDEX (DAX)	6278.90	63.65	1.02%	14:15
IBEX 35 INDEX (IBEX)	9616.20	-16.40	-0.17%	11:46
MILAN MIB30 INDEX (MIB30)	40220.00	429.00	1.08%	12:39
BEL20 INDEX (BEL20)	2820.99	18.64	0.67%	12:09
AMSTERDAM EXCHANGES INDX (AEX)	596.68	-1.12	-0.19%	11:00
SWISS MARKET INDEX (SMI)	7699.40	-6.40	-0.08%	5/23
Asia/Pacific				
Index	**Value**	**Chg**	**Pct Chg**	**Date**
NIKKEI 225 INDEX (NKY)	13895.79	-171.91	-1.22%	2:01
HANG SENG STOCK INDEX (HSI)	13810.60	-28.50	-0.21%	4:07
ASX ALL ORDINARIES INDX (AS30)	3350.50	-11.50	-0.34%	5/23
SING: STRAITS TIMES INDU (STI)	1692.38	5.42	0.32%	5:05

FIGURA 31 – FORMATAÇÃO EXPLÍCITA DA TABELA UTILIZADA NO EXEMPLO

Os dados relevantes aparecem nessa tabela formada por seis

grupos, respectivamente, três com a indicação do local (*North/Latin America, Europe/Africa, Asia/Pacific*) e três com os valores das ações. Esta única tabela deu origem a sete entidades no banco de dados: uma que agrupa todas as tabelas chamada **Doc_Tb1_Ln1_Cl1_Tb1_Cmp**, e outras seis tabelas cujos nomes têm a seguinte configuração: **Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt*i***, onde *i* representa a posição relativa da tabela no grupo.

As informações da página estão armazenadas no banco de dados, mas, como foi salientado, o fato de os *sites* se utilizarem extensamente do recurso tabela para formatação de suas páginas gera muitas entidades dispensáveis no banco de dados. Para encontrar a informação desejada, o usuário deve selecionar somente aquelas tabelas de interesse, a fim de realizar consultas SQL sobre os dados gerados. Elas são possíveis, como mostra o exemplo a seguir:

```
SELECT  Coluna0
FROM  Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt5
WHERE Data =(SELECT Max(Data)
                FROM Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt5)
      AND Value=(SELECT Max(Value)
                FROM Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt5);
```

Esta consulta devolve o nome da ação de maior valor na Ásia na última inclusão de dados realizada.

```
SELECT  Coluna0, Value
FROM  Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt3
WHERE Data = (SELECT Max(data)
                FROM Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt3)
      AND Hora = (SELECT Max(hora)
                FROM Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt3);
```

Esta consulta devolve o nome e o valor das ações das Américas na última inclusão de dados realizada.

```
SELECT T1.Coluna0, T2.Coluna0, T3.Coluna0
FROM ((Doc_Tb1_Ln1_Cl1_Tb1_Cmp T
      INNER JOIN Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt1 T1
        ON T.Chave = T1.ChaveEstrangeira)
      INNER JOIN Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt3 T2
```

```
   ON T.Chave = T2.ChaveEstrangeira)
INNER JOIN Doc_Tb1_Ln1_Cl1_Tb1_Cmp_TabInt5 T3
   ON T.Chave = T3.ChaveEstrangeira
WHERE   T.Data = (SELECT MIN(Data) FROM T)
AND T.Hora =     (SELECT MIN(Hora) FROM T)
AND T1.Value =    (SELECT MAX(Value)
                   FROM T1
                   WHERE T1.Data =
                      (SELECT MIN(Data) FROM T)
                   AND T1.Hora =
                      (SELECT MIN(Hora)FROM T))
AND T2.Value =   (SELECT MAX(Value)
                   FROM T2
                   WHERE T2.Data =
                      (SELECT MIN(Data) FROM T)
                   AND T2.Hora =
                      (SELECT MIN(Hora)FROM T))
AND T3.Value =   (SELECT MAX(Value)
                   FROM T3
                   WHERE T3.Data =
                      (SELECT MIN(Data) FROM T)
                   AND T3.Hora =
                      (SELECT MIN(Hora)FROM T));[11]
```

Essa consulta devolve o nome da ação de maior valor em cada uma das áreas na primeira pesquisa realizada. Aqui é possível verificar que o relacionamento entre as tabelas foi garantido por meio da tabela chamada Doc_Tb1_Ln1_Cl1_Tb1_Cmp.

Os *sites* que apresentam índices econômicos e financeiros representam aplicações potenciais do uso dessa tecnologia, já que, em geral, contêm informações numéricas no formato tabular e geram bases de dados ricas em tuplas.

Outro tipo de *site* que poderia ser utilizado com benefício é aquele que apresenta informações meteorológicas, como por exemplo, o que existe no endereço weather.noaa.gov. A página que apresenta informações climáticas da região de Curitiba foi também utilizada como exemplo durante os teste do protótipo.

---

[11] Nessa consulta foram utilizados *alias* para os nomes das tabelas a fim de facilitar a compreensão do comando. [17]

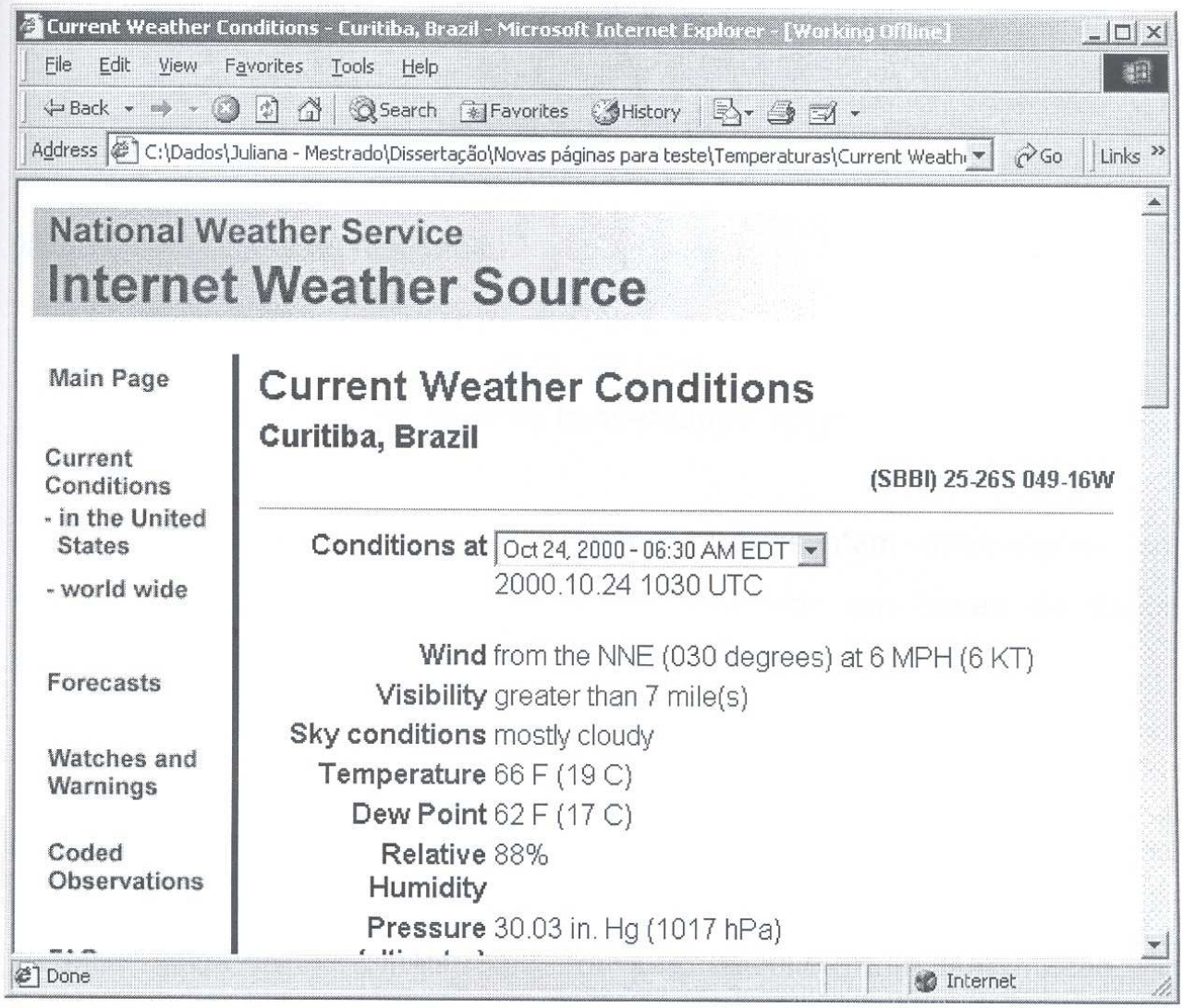


FIGURA 32 – PÁGINA COM INFORMAÇÕES METEOROLÓGICAS UTILIZADA COMO TESTE

Assim como aconteceu com a página da Bloomberg, a geração do banco de dados para este documento HTML deu origem a diversas relações vazias ou com dados irrelevantes do ponto de vista de extração de informações, pelo fato de terem o objetivo de formatação da página. As relações com informações relevantes têm os seguintes nomes:

- Doc_Tb1_Ln0_Cl3_Tb0_Cmp_TabInt1.
- Doc_Tb1_Ln0_Cl3_Tb1_Cmp_TabInt1.

Exemplos de consultas que poderiam ser feitas sobre essas relações são os seguintes:

```
SELECT Doc_Tb1_Ln0_Cl3_Tb0_Cmp_TabInt1.*
FROM Doc_Tb1_Ln0_Cl3_Tb0_Cmp_TabInt1
WHERE Doc_Tb1_Ln0_Cl3_Tb0_Cmp_TabInt1.Data=(SELECT
MAX(Data) FROM  Doc_Tb1_Ln0_Cl3_Tb0_Cmp_TabInt1);
```

Essa consulta mostra quais eram os indicadores meteorológicos na última coleta de dados realizada.

```
SELECT Coluna2
FROM Doc_Tb1_Ln0_Cl3_Tb1_Cmp_TabInt1
WHERE Data > #2/1/2001# AND Data < #3/1/2001#;
```

Essa pesquisa devolve as temperaturas observadas durante o mês de fevereiro de 2001.

Sites meteorológicos nem sempre apresentam informações em formato tabular, mas ter seu conteúdo preservado em bases de dados históricas pode ser interessante para uma aplicação de *data mining*.

Antes de encerrar este capítulo, é importante descrever algumas peculiaridades ocorridas durante a elaboração do trabalho.

Durante o desenvolvimento desse protótipo, algumas dificuldades foram encontradas. O uso de documentos da Internet se, por um lado, garantiu dados de teste mais reais, por outro dificultou a implementação. Foi necessária a implementação de um filtro que retira do documento todos os rótulos irrelevantes a fim de possibilitar a chegada a um resultado satisfatório.

Com isso, verificou-se que pode haver, na Internet, documentos com erro em relação à sintaxe HTML. Isso pode não gerar conseqüências para a apresentação da página, já que o *browser* a interpreta e apresenta um resultado, sem que o construtor da página possa perceber tal erro. Por exemplo, o seguinte trecho de código adaptado da página da Bloomberg utilizada nos testes do protótipo[12] contém um erro:

[12] Os atributos dos rótulos, bem como os tags não considerados na estruturação da página foram eliminados, o que acontece como resultado do processo de filtragem.

```
<TD>
   <TABLE>
      <TR>
         <TD>
            World Indices Tue, 29 May 2001, 3:48pm EDT
         </TD>
         <TD>
         </TD>
         <TD>
         </TD>
      </TR>
      </TD></TR>
      </TABLE>
</TD>
```

Existem dois rótulos em excesso, </TD> e </TR>, os quais aparecem destacados em negrito. É possível que a tabela contivesse mais uma linha, a qual foi parcialmente eliminada (apenas seus rótulos finais permaneceram). Como esta tabela está dentro de uma coluna de outra tabela, o tag </TD> acaba encerrando essa coluna, o que gera um erro de processamento. Para evitar esse tipo de erro seria necessário implementar uma espécie de verificador sintático do código HTML.

Outro problema ocorreu em relação ao domínio das colunas das tabelas. Supondo que em uma versão do documento houvesse uma tabela com a seguinte configuração:

Ação	Valor	Volume negociado
ABC PN	546,00	1000
XYZ ON	434,50	2000
ACS PN	137,34	570

Se, em outra versão do documento, a mesma tabela possuísse a seguinte configuração, um erro ocorreria na geração da base de dados:

Ação	Valor	Volume negociado
ABC PN	Não disp.	1000
XYZ ON	434,50	2000
ACS PN	137,34	570

A criação da base de dados para a primeira tabela identifica como *real* o domínio da coluna **Valor**. Já na segunda tabela, a mesma coluna possui domínio *caractere*, pelo fato de haver palavras na primeira linha no lugar do valor.

O comando SQL para inclusão de dados na segunda tabela o fará considerando os valores como caracteres, o que será incompatível com o tipo de dados do comando CREATE TABLE, o qual é gerado com base na primeira tabela. Este é um problema que ainda não foi corrigido.

Embora tenham sido descritos problemas apresentados no protótipo, os dois fatos que deram origem a essas dificuldades ocorreram apenas uma vez, e na mesma página utilizada para testes. A solução desses problemas pode fazer parte do desenvolvimento futuro desse projeto.

Este capítulo discutiu os resultados obtidos com o processamento de páginas fictícias e reais, além de alguns problemas ocorridos durante o uso de páginas reais para teste. Observou-se que efetivamente o conteúdo dos documentos HTML fica armazenado em banco de dados relacional, como era o objetivo deste trabalho. Entretanto, rótulos de formatação acabam prejudicando o resultado final, em relação à compreensão da base gerada, já que dão origem a diversas tabelas vazias ou cujos campos são nulos.

Esse fato, no entanto, não impede que consultas SQL possam ser feitas sobre o banco de dados criado. As diversas consultas SQL efetuadas sobre as bases geradas confirmaram a efetividade e a utilidade da proposta.

5 CONCLUSÕES E TRABALHOS FUTUROS

No início do desenvolvimento deste trabalho verificou-se que as pesquisas em relação a dados semi-estruturados atuam em uma ou mais das seguintes direções: modelagem de dados semi-estruturados, linguagem de consulta a dados semi-estruturados e inferência de estrutura em dados semi-estruturados.

Observou-se que o modelo mais utilizado para representação de dados semi-estruturados é o modelo OEM [26], no qual os dados são auto-descritos, ou seja, podem ser compreendidos sem que seja necessária nenhuma referência a qualquer esquema externo.

Com relação às linguagens de consulta foram identificadas duas abordagens possíveis: acrescentar funcionalidades à linguagem SQL para que ela seja capaz de acessar dados semi-estruturados [24] ou criar uma linguagem baseada nas noções de dados semi-estruturados para depois ajustá-la a questões relativas a performance e sintaxe [26][8][15][13].

Por fim, em relação à inferência de uma possível estrutura que exista por trás dos dados semi-estruturados, foram relacionados os projetos TSIMMIS[26] e NoDoSe [4]. O primeiro extrai informações de documentos HTML e gera objetos segundo o modelo OEM. O segundo determina de maneira semi-automática a estrutura de um documento e extrai seus dados.

Com base no estudo dessas pesquisas decidiu-se o objetivo deste trabalho que era o de inferir um esquema de banco de dados relacional a partir de dados semi-estruturados, representados por um documento HTML.

O acompanhamento do desenvolvimento das pesquisas em relação a dados semi-estruturados levou à identificação de alguns trabalhos que caminharam no mesmo sentido, qual seja, o de armazenar o conteúdo de documentos semi-estruturados em bases de dados relacionais. Dois trabalhos devem ser citados.

Em [30] e [31] é descrita uma forma de armazenar o conteúdo de um documento semi-estruturado escrito em linguagem XML em uma base de dados relacional. O documento, primeiramente é estruturado em termos de objetos OEM. Depois esses objetos são analisados e mapeados em relações com base nos atributos que possuem. Isso é feito por uma linguagem chamada STORED.

O segundo trabalho, descrito em [32], infere a estrutura de um documento HTML e apresenta essa estrutura para o usuário. Este atribui valor semântico aos atributos encontrados no documento observando o conteúdo real das páginas no *site* e decide quais deles devem ser armazenados em relações.

Em relação ao objetivo inicial deste trabalho, verificou-se que é possível inferir a estrutura de um documento HTML e armazenar seu conteúdo em um banco de dados relacional, o que foi mostrado pelo protótipo implementado. Ele efetivamente cria um banco de dados relacional com base numa página da Internet.

Entretanto, ainda outros caminhos se abrem e outras possibilidades de continuação deste trabalho se apresentam.

O protótipo cria a base de dados, levando em consideração a estrutura interna de um documento HTML. No entanto, essa estrutura muitas vezes não é visível ao usuário de uma página, e tem a função exclusiva de melhorar a formatação do documento. O fato de ela ficar explicitada na base de dados pode vir a confundir mais do que a auxiliar o usuário de um banco de dados criado com base na utilização dessa tecnologia. Um direcionamento fundamental para melhorar a qualidade do banco de dados obtido diz respeito à filtragem da base de dados gerada, eliminando do esquema de banco de dados aquelas relações que estivessem vazias.

Além disso, duas outras possibilidades se abrem:

- Permitir que o próprio usuário do sistema escolha como a base de dados será criada, incluindo um passo intermediário entre o reconhecimento da estrutura de uma página e a geração dos *scripts* de banco de dados, de maneira parcialmente análoga ao que é feito no projeto descrito em [32];
- Determinar heurísticas que possam fazer essa análise.

Na primeira alternativa, a estrutura da página seria mostrada ao usuário que poderia escolher, então, quais níveis de hierarquia e quais rótulos deveriam ser considerados na criação da base de dados. Isso tornaria o resultado final provavelmente mais direcionado às expectativas do usuário, mas faria com que o processo, hoje automático, perdesse essa característica e se tornasse semi-automático.

A segunda alternativa manteria a característica de automação, por meio da implementação de novas heurísticas que viessem a tornar o resultado mais significativo para o usuário.

Outra possibilidade de futuro para essa tecnologia é a troca da linguagem utilizada como base para documentos semi-estruturados. O trabalho foi feito sobre linguagem HTML, mas, atualmente, a linguagem XML concentra a maioria dos esforços de pesquisa quando se fala em documentos para Internet. É bastante possível que a linguagem XML forneça resultados mais significativos, já que um dos recursos disponível diz respeito à possibilidade de definição de estruturas de informação [21][22]. Essa definição de estrutura poderia ser aproveitada na inferência do que é relevante em um documento.

Os itens discutidos anteriormente dizem respeito a trabalhos futuros desenvolvidos sobre a própria tecnologia pesquisada. Entretanto, outra possibilidade de trabalho futuro diz respeito à manipulação da base de dados

em si. Aplicações de *data mining* poderiam ser desenvolvidas sobre as bases de dados geradas, fechando o ciclo que se inicia com o interesse do usuário por uma determinada página, a criação de um banco de dados temporal sobre o conteúdo dessa página e a posterior manipulação desses dados de forma a produzir o resultado esperado por este usuário.

Por fim, verificou-se que é possível a extração automática de informações de uma base de dados semi-estruturados e posterior criação de uma base de dados estruturados. Esse era o objetivo inicial deste trabalho e ele foi atingido.

Aplicações em potencial para esta tecnologia existem e já foram citadas (dados meteorológicos e de mercado financeiro). Entretanto, essas são somente as aplicações mais óbvias. Conforme demonstrado, qualquer *site* desenvolvido em linguagem HTML pode ser mapeado em um banco de dados relacional. Um requisito para que essa tecnologia seja de utilidade é que o *site* tenha seu conteúdo atualizado periodicamente, a fim de que a base de dados possa ter quantidade significativa de tuplas.

Em relação aos outros trabalhos pesquisados, a vantagem da proposta apresentada reside no caráter automático da extração de conteúdo. O usuário não precisa ter qualquer conhecimento *a priori* do código HTML a fim de que os dados da página sejam armazenados em relações de um banco de dados.

Este foi o primeiro passo, agora resta o desenvolvimento dessa técnica com o objetivo de melhorar ainda o mais o resultado alcançado.

REFERÊNCIAS BIBLIOGRÁFICAS

1 _____. **Arquivos de ajuda do Microsoft Access 97.**

2 ABITEBOUL, S., BUNEMAN, P., SUCIU, D. **Data on the *Web***: from relations to semistructured data and XML. Sao Francisco: Morgan Kaufmann, 2000. 258 p.

3 ABITEBOUL, Serge et al. **The Lorel query language for semistructured data**. In International Journal on Digital Libraries (JODL 97). Disponível em osage.inria.fr/verso/PUBLI/all-bykey.php?mytexte=abiteboul. Acesso 29 mai. 2001.

4 ADELBERG, B. **NoDoSe**: a tool for semi-automatically extracting structured and semistructured data from text-documents. In SIGMOD 1998. Disponível em www.cs.northwestern.edu/~adelberg/. Acesso 28 mai. 2001.

5 BATINI, C., CERI, S. NAVATHE, S. B. **Conceptual database design:** an entity-relationship approach. Redwood City: Benjamin/Cummings, 1992. 496 p.

6 BOOCH, R.; RUMBAUGH,J.; JACOBSON,I. **The unified modeling language user guide**. Reading: Addison-Wesley, 1998. 482 p.

7 BOYAN, J. FREITAG, D., JOACHIMS, T. **A machine learning architecture for optimizing web search engines**. Technical Report WS-96-05, American Association of Artificial Intelligence, 1994. Disponível em: <citeseer.nj.nec.com/boyan96machine.html>. Acesso 01 ago. 2001.

8 BUNEMAN, P., DAVIDSON, S., HILLEBRAND, G. **A query language and optimization techniques for unstructured data**. In Proceedings of ACMSIGMOD International Conference on Management of Data, pages 505--516, Montreal, Canada, June 1996. Disponível em <citeseer.nj.nec.com/buneman96query.html>. Acesso 29 mai. 2001.

9 COOLEY, R. MOBASHER B. , SRIVASTAVA, J. **Web Mining:** information and pattern discovery on the World Wide Web. In Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97), November 1997. Disponível em: <citeseer.nj.nec.com/cooley97web.html>. Acesso 31 ago. 2001.

10 DATE, C. J. **Introdução a Sistemas de Bancos de Dados**. *Tradução da 7ª edição americana*. Rio de Janeiro: Campus, 2000. 803 p.

11 DATE, C. J., DARWEN, H. **Foundation for objetc/relational databases**: the third manifesto. Reading, Addison Wealey, 1998. 495 p.

12 EDELWEISS, N. Bancos de Dados Temporais: Teoria e Prática. In: Jornada de Atualização em Informática, 17., 1998, Recife. **Anais do XVIII Congresso Nacional da Sociedade Brasileira de Computação "Rumo à Sociedade do Conhecimento".** Recife: Sociedade Brasileira de Computação, 1998. v. 2. p.225-282.

13 FERNANDEZ, M. et al. **Cathing the Boat with Strudel**: Experiences with a Web-Site Management System. In SIGMOD , 1998. Disponível em:

<www.research.att.com/~mff/ HomePage.genoid_34.html>. Acesso em 30 jan. 2001

14 FIEBIG, T., WEISS, J., MOERKOTTE, G. **RAW:** A Relational Algebra for the Web. In Proceedings of the Workshop on Management of Semistructured Data, Tucson, Arizona, May 1997. Disponível em citeseer.nj.nec.com/fiebig97raw.html. Acesso 29 mai. 2001.

15 HAMMER, J. et al. **Information translation, mediation and mosaic-based browsing in the TSIMMIS system**. In Exhibits Program of the Proceedings of the ACM SIGMOD International Conference on Management of Data, page 483, San Jose, California, June 1995. Disponível em www-db.stanford.edu/tsimmis/publications.html. Acesso 28 mai. 2001.

16 HEUSER, C. A.; DORNELES, C. F. e NORONHA, M. A. **Ligando a Tecnologia de Banco de Dados com a Gestão de Documentos**. In: ISDM 98 – Seminário Internacional de Gestão de Documentos, 1998, Curitiba. Anais. Curitiba : [?], 1998. CD-ROM.

17 HURSCH, C. J., HURSCH, J. L. **SQL linguagem de consulta estruturada**. Rio de Janeiro: LTC, 1990. 181 p.

18 ISO **International Organization for Standartization**. <www.iso.ch/welcome.html>. Acesso em 06 mai. 2001.

19 JOACHIMS, T. et al. **WebWatcher:** machine learning and hypertext. In Beiträge zum 7. Fachgruppentreffen MASCHINELLES LERNEN der GI-Fachgruppe 1.1.3, 1995, Forschungsbericht Nr. 580 der Universität Dortmund. Disponível em: <www-ai.cs.uni-dortmund.de/PERSONAL/joachims.html>. Acesso em 04 jun. 2001.

20 KONOPNICKI, D. e SHMUELI, O. **W3QS**: a query system for the World-Wide Web. In VLDB'95. Disponível em www.cs.technion.ac.il/~konop/w3qs.html#publications. Acesso 29 mai. 2001.

21 LIGHT, R. **Iniciando em XML**. São Paulo: Makron Books, 1999. 404 p.

22 MECCA, G. e ATZENI, P. **Cut and Paste**. In Journal of Computing and System Sciences, Special Issue on PODS'97, 1999. Disponível em <www.dia.uniroma3.it/Araneus/articles.html> Acesso em 30 jan. 2001

23 Microsoft Corporation. **MSDN Library 2000**. CD-ROM.

24 MIHAILA, G. A. **WebSQL – An SQL-like query language for the World Wide Web**. MSc. Thesis, University of Toronto, 1996. Disponível em www.cs.toronto.edu/~georgem/. Acesso 29 mai. 2001.

25 MLADENIC, D. **Personal WebWatcher:** design and implementation. Technical Report IJS-DP-7472, School of Computer Science, Carnegie-Mellon University, Pittsburgh, USA, October. Disponível em: <citeseer.nj.nec.com/mladenic96personal.html>. Acesso 01 ago. 2001.

26 MOLINA. G. M et al. **The TSIMMIS approach to mediation**: data models and languages. In Journal of Intelligent Information Systems, 1997. Disponível em www-db.stanford.edu/tsimmis/publications.html. Acesso em 28 mai. 2001.

27 OMG **Objetc Management Group**. <www.omg.org> Acesso em 06. Ago. 2001.

28 TITTEL, E.; STEWART, J. M; PITTS, N. **HTML 4**. São Paulo: Berkeley Brasil, 1998. 230 p.

29 W3C. **HTML 4.0 Specification**. Disponível em: <www.w3.org/TR/1998/REC-html40-19980424>. Acesso em 01 fev. 2001.

30 DEUTSCH, A., FERNANDEZ, M., SUCIU, D. **Storing semistructured data in relations**. In Workshop on "Query Languages for semistructured data and non-standard data formats" ICDT'99, Jerusalem, January 1999. Disponível em: <http://citeseer.nj.nec.com/200178.html>. Acesso em 01 ago. 2001.

31 DEUTSCH, A., FERNANDEZ, M., SUCIU, D. **Storing semistructured data whith STORED**. Disponível em: <www.research.att.com/~mff/HomePage.genoid_34.html>. Acesso em 01 ago. 2001.

32 GRUMBACH, S., MECCA, G. **In search of the lost schema**. Disponível em: <citeseer.nj.nec.com/grumbach99search.html>. Acesso 31 ago. 2001.

REFERÊNCIAS COMPLEMENTARES

ABITEBOUL, Serge et al. **Views for semistructured data**. Disponível em osage.inria.fr/verso/PUBLI/all-bykey.php?mytexte=abiteboul. Acesso 29 mai. 2001.

ABITEBOUL, Serge. **Querying semistructured data**. Disponível em osage.inria.fr/verso/PUBLI/all-bykey.php?mytexte=abiteboul. Acesso 29 mai. 2001.

ADALI, S. et al. **Query caching and optimization in distributed mediator systems**. Disponível em www.cs.rpi.edu/~sibel/research/publications.html. Acesso 29 mai. 2001.

ASHISH, N., KNOBLOCK, C. **Wrapper generation for semistructured Internet sources**. Disponível em www.isi.edu/sims/naveen/pub.html. Acesso 29 mai. 2001.

ATZENI, P.; MECCA, G., MERIALDO, P. **Semistructured and structured data in the Web:** going back and forth. Disponível em citeseer.nj.nec.com/atzeni97semistructured.html. Acesso 29 mai. 2001.

BAYARDO, R. J. et al. **InfoSleuth:** semantic integration of information in open and dynamic environments. Disponível em www.almaden.ibm.com/cs/people/bayardo/papers.html. Acesso 29 mai. 2001.

BLAKE, G. E. et al. **Text/relational database management systems:** overview and proposed SQL extensions. 1995. Disponível em db.uwaterloo.ca/OED/trdbms.html. Acesso 29 mai. 2001.

BUNEMAN, P., DAVIDSON, S., FERNANDEZ, M. e SUCIU, D. **Adding structure to unstructured data**. Disponível em citeseer.nj.nec.com/buneman97adding.html. Acesso 29 mai. 2001.

CANDAN, K. S., JAJODIA, S., SUBRAHMANIAN, V. S. **Secure mediated databases**. Disponível em ise.gmu.edu/~csis/faculty/publications.html. Acesso 29 mai. 2001.

CERÍCOLA, O. V. **Banco de Dados Relacional e Distribuído**. Rio de Janeiro: LTC, 1991. 335 p.

CHAWATHE, S., ABITEBOUL, S., WIDOM, J. **Representing and querying changes in semistructured data**. Disponível em www.cs.umd.edu/~chaw/pubs/. Acesso 29 mai. 2001.

CLUET, S. et al. **Your mediators need data conversion!** Disponível em cosmos.inria.fr:8080/cgi-bin/publisverso?what=byyear2&query=cluet. Acesso 29 mai. 2001.

ERIKSSON, H. , PENKER, M. **UML Toolkit**. New York: Wiley, 1998. 397 p.

FERNANDEZ, M., SUCIU, D. **Optimizing regular path expressions using graph schemas**. Disponível em www.research.att.com/~mff/HomePage.genoid_34.html. Acesso 29 mai. 2001.

FERNANDEZ, Mary et al. **Cathing the boat with Strudel:** experiences with a website management system. Disponível em www.research.att.com/~mff/HomePage.genoid_34.html. Acesso 29 mai. 2001.

FERNANDEZ, Mary et al. **Reasoning about website structure**. Disponível em www-caravel.inria.fr/Fmbrepubs_dana.html. Acesso 29 mai. 2001.

FLORESCU, D., LEVY, A., MENDELZON, A. **Database techniques for the World Wide Web:** a survey. Disponível em www-caravel.inria.fr/Fmbrepubs_dana.html. Acesso 29 mai. 2001.

GARCIA-MOLINA, H. et al. **Integrating and accessing heterogeneous information sources in TSIMMIS**. Disponível em www-db.stanford.edu/tsimmis/publications.html. Acesso 29 mai. 2001.

GOLDMAN, R. et al. **A standard textual interchange format for the Object Exchange Model (OEM)**. Disponível em www-db.stanford.edu/~mchughj/oemsyntax/oemsyntax.html. Acesso 29 mai. 2001.

HAMMER, J. et al. **Extracting semistructured information from the web**. Disponível em citeseer.nj.nec.com/hammer97extracting.html. Acesso 29 mai. 2001.

HASS, L. M. et al. **Optimizing queries across diverse data sources**. Disponível em citeseer.nj.nec.com/haas97optimizing.html. Acesso 29 mai. 2001.

HULL, R. **Managing semantic heterogeneity in databases**: a theoretical perspective. Disponível em citeseer.nj.nec.com/hull97managing.html. Acesso 29 mai. 2001.

JACOBSON, I. et al. **Object-Oriented Software Engineering:** a use case driven approach. Harlow: Addison-Wesley, 1997.

KASHYAP, V., RUSINKIEWICZ. **Modeling and querying textual data using E-R models and SQL**. Disponível em citeseer.nj.nec.com/kashyap97modeling.html. Acesso 29 mai. 2001.

KUSHMERICK, N., WELD, D. S., DOORENBOS, R. **Wrapper induction for information extraction**. Disponível em www.cs.ucd.ie/staff/nick/home/research/wrappers/. Acesso 29 mai. 2001.

MECCA, G. e ATZENI, P. **Cut and Paste**. Disponível em citeseer.nj.nec.com/mecca98cut.html. Acesso 29 mai. 2001.

MECCA, G. et al. **The ARANEUS web-based management system**. Disponível em citeseer.nj.nec.com/9692.html. Acesso 29 mai. 2001.

MENDELZON, A. O., MIHAILA, G. A., MILO, T. **Querying the World Wide Web**. Disponível em www.cs.toronto.edu/~mendel/papers.html. Acesso 29 mai. 2001.

NESTOROV, S. et al. **Representative objects:** concise representations of semistructured, hierarchical data. Disponível em www-db.stanford.edu/people/evtimov.html. Acesso 29 mai. 2001.

NESTOROV, S., ABITEBOUL, S., MOTWANI, R. **Extracting schema from semistructured data**. Disponível em www-db.stanford.edu/people/evtimov.html. Acesso 29 mai. 2001.

NESTOROV, S., ABITEBOUL, S., MOTWANI, R. **Inferring structure in semistructured data.** Disponível em www-db.stanford.edu/people/evtimov.html. Acesso 29 mai. 2001.

OLIBONI, B. QUINTARELLI, E., TANCA, L. **Temporal aspects of semistructured data**. In *Proceedings of The Eighth International Symposium on Temporal Representation and Reasoning (TIME-01)*, 2001. Disponível em <www.elet.polimi.it/Users/DEI/Sections/Compeng/Barbara.Oliboni/papers.html>. Acesso em 30 jul. 2001.

PAPAKONSTANTINOU, Y., GUPTA, A., HAAS, L. **Capabilities-based query rewriting in mediator systems**. Disponível em www.db.ucsd.edu/people/yannis.htm. Acesso 29 mai. 2001.

ROTH, M. T., SCHWARZ, P. **Don't Scrap It, Wrap It! A Wrapper Architecture for Legacy Data Sources**. Disponível em citeseer.nj.nec.com/roth97wrapper.html. Acesso 29 mai. 2001.

SUBRAHMANIAN, V. S. et al. **HERMES**: A Heterogeneous Reasoning and Mediator System. Disponível em www.cs.umd.edu/projects/hermes/publications/abstracts/hermes.html. Acesso 29 mai. 2001.

SUCIU, D. **An Overview of Semistructured Data**. Disponível em www.cs.washington.edu/homes/suciu/NodeInternal,sitegraph.genoid_1.html. Acesso 29 mai. 2001.

ANEXO 1 – DESCRIÇÃO DOS MÉTODOS DOS OBJETOS QUE COMPÕEM A ESTRUTURA DE UM DOCUMENTO

Para cada um dos objetos gerados com base no código HTML de uma página existem alguns métodos que são relevantes. Este anexo tem por objetivo dar uma explicação breve desses métodos.

Classe	Método	Objetivo do método
CITag	criaTabela	Método polimórfico que grava o(s) comando(s) SQL necessário(s) para a criação das tabelas de bancos de dados. A implementação desse método é diferente em cada classe derivada de CITag.
	insereDados	Método polimórfico que grava o(s) comando(s) SQL necessário(s) para a inclusão de dados nas tabelas de bancos de dados. A implementação desse método é diferente em cada classe derivada de CITag.
CICorpo	montaEstrutura	Método que cria a estrutura de objetos de um documento ou de um elemento de um documento. Pode ser chamado recursivamente a partir de outros rótulos se eles contiverem seus próprios corpos.
	criaBDCorpo	Grava o(s) comando(s) SQL necessários para a criação das tabelas de bancos de dados para os objetos que possuem corpos.
	atualizaBDCorpo	Grava o(s) comando(s) SQL necessários para a inclusão de dados nas tabelas de bancos de dados para os objetos que possuem corpos.
CICelula	haTagsInternos	Verifica se uma célula possui rótulos internos, o que caracteriza a presença de um corpo.
	montaCelula	Extrai o conteúdo de uma célula que não possua corpo, ou cria o corpo de uma célula se esta o possuir.
	tipoDado	Devolve o tipo de dado associado a uma célula.
CILinhaTb	montaLinha	Monta a linha de uma tabela, reconhecendo e montando cada uma de suas células.
CITabela	montaTabela	Monta uma tabela reconhecendo cada uma de suas linhas.
	criaVetorGrupos	Cria um vetor que representa quantas colunas tem cada agrupamento em uma tabela. Um grupo é delimitado por um número de linhas consecutivas que possui a mesma quantidade de colunas.
	criaMatrizCorpo	Cria uma matriz que indica em que posição grupo, linha, coluna de uma matriz estão as células que possuem corpo interno.
	nomeColuna	Devolve uma *string* identificando qual deve ser o nome da coluna no banco de dados.

Classe	Método	Objetivo do método
ClParagrafo	haTagsInternos	Verifica se um parágrafo possui rótulos internos, o que caracteriza a presença de um corpo.
	extraiConteudo	Extrai o conteúdo textual de um parágrafo.
	montaParagrafo	Monta um parágrafo extraindo seu conteúdo ou construindo seu corpo, caso ele exista.
ClDocumento	montaEstrutura	Extrai todas as informações relevantes de um documento e ordena a criação de seu corpo.
	filtraHtml	Retira do código HTML todos os rótulos que não são relevantes para o reconhecimento de sua estrutura.
ClListaSimples	montaLista	Monta uma lista simples reconhecendo cada uma de suas linhas.
ClItemLista	haTagsInternos	Verifica se a descrição de uma lista possui um corpo.
	montaItem	Reconhece e extrai o conteúdo dos componentes de uma lista de definição.
ClListaDefinicao	montaLista	Verifica quantos são as linhas de uma lista de definição e monta cada uma delas.
ClPreFormatado	extraiTexto	Extrai o conteúdo textual de um texto pré-formatado.
	formaTabela	Verifica se o conteúdo de um texto pré-formatado pode formar uma tabela.
	montaTabela	Inclui os rótulos HTML necessários para criar uma tabela a partir de um texto pré-formatado.

ANEXO 2 – CÓDIGO HTML DOS EXEMPLOS UTILIZADOS NA GERAÇÃO DA BASE DE DADOS

Código da primeira versão:

```
<html>
<body>

<p>Este é um parágrafo de texto normal. O próximo
contém um corpo interno:</p>

<p>
   <cite>Uma citação.</cite>
   <code>Uma linha de código</code>
   <cite>Outra citação.</cite>
</p>

<div>
   <cite>
      <samp> Uma saída dentro de uma citação, que está
             inserida em uma outra hierarquia: uma
             divisão.
      </samp>
   </cite>
</div>

<table>
  <tr>
    <td>Célula 1</td>
  </tr>
  <tr>
    <td>Célula 2</td>
  </tr>
  <tr>
    <td>Grupo 2.1</td>
    <td>Grupo 2.2</td>
  </tr>
  <tr>
    <td>01/01/2001</td>
    <td>5</td>
  </tr>
  <tr>
    <td>04/02/2001</td>
    <td>9</td>
  </tr>
</table>
```

```
<p>A próxima tabela tem uma tabela interna:</p>

<table>
  <tr>
    <td>Coluna 1</td>
  </tr>
  <tr>
    <td><table>
      <tr>
        <td>01/01/2001</td>
        <td>2.34</td>
      </tr>
      <tr>
        <td>02/04/2001</td>
        <td>3.56</td>
      </tr>
    </table>
    </td>
  </tr>
</table>

<p>Exemplo de lista simples:

<ul>
  <li>Primeira linha</li>
  <li>Segunda linha</li>
  <li>Terceira linha</li>
</ul>

<p>Exemplo de lista de definição:

<dl>
  <dt>Termo 1 </dt>
  <dd>Este termo é simples </dd>
  <dt>Termo 2 </dt>
  <dd>  <p>Este termo tem um corpo </p>
        <cite>Uma citação</cite>
        <div>Uma divisão </div>
  </dd>
</dl>
</body>
</html>
```

Código da segunda versão:

```
<html>
<body>

<p>Este é um parágrafo de texto normal. O próximo
contém um corpo interno (versão 2):</p>
```

```
<p>
   <cite>Uma citação (v2).</cite>
   <code>Uma linha de código (v2)</code>
   <cite>Outra citação (v2).</cite>
</p>

<div>
   <cite>
     <samp>  Uma saída dentro de uma citação, que está
             inserida em uma outra hierarquia: uma
             divisão
     </samp>
   </cite>
</div>

<table>
  <tr>
    <td>Célula 1 (v2)</td>
  </tr>
  <tr>
    <td>Célula 2 (v2)</td>
  </tr>
  <tr>
    <td>Célula 3 (v2)</td>
  </tr>
  <tr>
    <td>Grupo 2.1 (v2)</td>
    <td>Grupo 2.2 (v2)</td>
  </tr>
  <tr>
    <td>01/08/2001</td>
    <td>51</td>
  </tr>
  <tr>
    <td>04/09/2001</td>
    <td>97</td>
  </tr>
  <tr>
    <td>05/05/2001</td>
    <td>34</td>
  </tr>
</table>

<p>A próxima tabela tem uma tabela interna: (v2)</p>

<table>
  <tr>
    <td>Coluna 1 (v2)</td>
  </tr>
```

```
  <tr>
    <td><table>
      <tr>
        <td>01/08/2001</td>
        <td>2.34</td>
      </tr>
      <tr>
        <td>03/03/2001</td>
        <td>2</td>
      </tr>
      <tr>
        <td>02/09/2001</td>
        <td>3.56</td>
      </tr>
    </table>
    </td>
  </tr>
</table>

<p>Exemplo de lista simples: (v2)

<ul>
  <li>Primeira linha (v2)</li>
  <li>Segunda linha (v2)</li>
  <li>Terceira linha (v2)</li>
  <li>Quarta linha (v2)</li>
</ul>

<p>Exemplo de lista de definição: (v2)

<dl>
  <dt>Termo 1 (v2)</dt>
  <dd>Este termo é simples (v2)</dd>
  <dt>Termo 2  (v2)</dt>
  <dd> <p> Este termo tem um corpo (v2)</p>
      <cite>Uma citação (v2)</cite>
      <div>Uma divisão (v2)</div>
  </dd>
</dl>
</body>
</html>
```

ANEXO 3 – *SCRIPT* DE BANCO DE DADOS GERADO PELO PROCESSAMENTO DE DUAS VERSÕES DE UM DOCUMENTO

Código DDL

```
CREATE TABLE Doc_Pg0 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);

CREATE TABLE Doc_Pg1 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Data DATE, Hora TIME);

CREATE TABLE Doc_Pg1_Corpo (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME, ChaveEstrangeira INTEGER CONSTRAINT
Doc_Pg1_Corpo REFERENCES Doc_Pg1);[13]

CREATE TABLE Doc_Dv2 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Data DATE, Hora TIME);

CREATE TABLE Doc_Dv2_Ct0 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Data DATE, Hora TIME,
ChaveEstrangeira INTEGER CONSTRAINT Doc_Dv2_Ct0
REFERENCES Doc_Dv2);

CREATE TABLE Doc_Dv2_Ct0_Corpo (Chave INTEGER
CONSTRAINT Chave_Primaria PRIMARY KEY, Texto LONGTEXT,
Data DATE, Hora TIME, ChaveEstrangeira INTEGER
CONSTRAINT Doc_Dv2_Ct0_Corpo REFERENCES Doc_Dv2_Ct0);

CREATE TABLE Doc_Tb3_Cmp (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Data DATE, Hora Time);

CREATE TABLE Doc_Tb3_Cmp_TabInt0 (Chave INTEGER
CONSTRAINT Chave_Primaria PRIMARY KEY, [Coluna0]
LONGTEXT,  Data DATE, Hora TIME, ChaveEstrangeira
INTEGER CONSTRAINT Doc_Tb3_Cmp_TabInt0 REFERENCES
Doc_Tb3_Cmp);

CREATE TABLE Doc_Tb3_Cmp_TabInt1 (Chave INTEGER
CONSTRAINT Chave_Primaria PRIMARY KEY, [Grupo 2 1]
DATETIME, [Grupo 2 2]  INTEGER,  Data DATE, Hora TIME,
ChaveEstrangeira INTEGER CONSTRAINT Doc_Tb3_Cmp_TabInt1
REFERENCES Doc_Tb3_Cmp);
```

[13] A instrução CONSTRAINT *nome* REFERENCES *relaçãoexterna* estabelece um relacionamento 1:N entre a tabela que está sendo criada e aquela referenciada como *relaçãoexterna*. Isto impõe integridade referencial, entretanto, essa cláusula não estabelece alterações e inclusões em cascata, ou seja, uma tupla não poderá ser excluída de uma relação se tiver relacionamentos com outras relações [1].

```
CREATE TABLE Doc_Pg4 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);

CREATE TABLE Doc_Tb5 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, [Coluna 1]  LONGTEXT,  Data
DATE, Hora TIME);

CREATE TABLE Doc_Tb5_Ln1_Cl0_Tb0 (Chave INTEGER
CONSTRAINT Chave_Primaria PRIMARY KEY, [Coluna0]
DATETIME, [Coluna1]  DOUBLE,  Data DATE, Hora TIME);

CREATE TABLE Doc_Pg6 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);

CREATE TABLE Doc_ListaSimples7 (Chave INTEGER
CONSTRAINT Chave_Primaria PRIMARY KEY, Linha LONGTEXT,
Data DATE, Hora TIME);

CREATE TABLE Doc_Pg8 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);

CREATE TABLE Doc_LD9 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Definicao LONGTEXT,
Descricao LONGTEXT, Data DATE, Hora TIME);

CREATE TABLE Doc_LD9_Pg0 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);

CREATE TABLE Doc_LD9_Ct1 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);

CREATE TABLE Doc_LD9_Dv2 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);

CREATE TABLE Doc_Tb5_Ln1_Cl0_Tb0 (Chave INTEGER
CONSTRAINT Chave_Primaria PRIMARY KEY, [Coluna0]
DATETIME, [Coluna1]  DOUBLE,  Data DATE, Hora TIME);

CREATE TABLE Doc_LD9_Pg0 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);

CREATE TABLE Doc_LD9_Ct1 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);

CREATE TABLE Doc_LD9_Dv2 (Chave INTEGER CONSTRAINT
Chave_Primaria PRIMARY KEY, Texto LONGTEXT, Data DATE,
Hora TIME);
```

Código DML:

```
SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg0 (Chave, texto, data, hora) VALUES
(ChavePrimaria, 'Este é um parágrafo de texto normal. O
próximo contém um corpo interno: ', '21/8/2001',
'15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg1 (Chave, data, hora) VALUES
(ChavePrimaria, '21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1_Corpo;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Pg1_Corpo (Chave, texto, data, hora,
ChaveEstrangeira) VALUES (ChavePrimaria, 'Uma citação.
', '21/8/2001', '15:33:20', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1_Corpo;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Pg1_Corpo (Chave, texto, data, hora,
ChaveEstrangeira) VALUES (ChavePrimaria, 'Uma linha de
código ', '21/8/2001', '15:33:20', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1_Corpo;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Pg1_Corpo (Chave, texto, data, hora,
ChaveEstrangeira) VALUES (ChavePrimaria, 'Outra
citação. ', '21/8/2001', '15:33:20', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM Doc_Dv2;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Dv2 (Chave, data, hora) VALUES
(ChavePrimaria, '21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Dv2_Ct0;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Dv2;
```

```
PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Dv2_Ct0 (Chave, data,
hora,ChaveEstrangeira) VALUES (ChavePrimaria,
'21/8/2001', '15:33:20', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Dv2_Ct0_Corpo;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Dv2_Ct0;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Dv2_Ct0_Corpo (Chave, texto, data,
hora, ChaveEstrangeira) VALUES (ChavePrimaria, 'Uma
saída dentro de uma citação, que está inserida em uma
outra hierarquia: uma divisão. ', '21/8/2001',
'15:33:20', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Tb3_Cmp (Chave, Data, Hora) VALUES
(ChavePrimaria, '21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt0;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Tb3_Cmp_TabInt0 (Chave, [Coluna0],
Data, Hora, ChaveEstrangeira) VALUES (ChavePrimaria,
'Célula 1', '21/8/2001', '15:33:20', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt0;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Tb3_Cmp_TabInt0 (Chave, [Coluna0],
Data, Hora, ChaveEstrangeira) VALUES (ChavePrimaria,
'Célula 2', '21/8/2001', '15:33:20', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt1;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;
```

```
INSERT INTO Doc_Tb3_Cmp_TabInt1 (Chave, [Grupo 2 1],
[Grupo 2 2], Data, Hora, ChaveEstrangeira) VALUES
(ChavePrimaria, '01/01/2001', 5, '21/8/2001',
'15:33:20', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt1;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Tb3_Cmp_TabInt1 (Chave, [Grupo 2 1],
[Grupo 2 2], Data, Hora, ChaveEstrangeira) VALUES
(ChavePrimaria, '04/02/2001', 9, '21/8/2001',
'15:33:20', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg4;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg4 (Chave, texto, data, hora) VALUES
(ChavePrimaria, 'A próxima tabela tem uma tabela
interna: ', '21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb5;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Tb5 (Chave, [Coluna 1], Data, Hora)
VALUES (ChavePrimaria, 'Possui corpo. Tabelas começam
com Doc_Tb5_Ln1_Cl0', '21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb5_Ln1_Cl0_Tb0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Tb5_Ln1_Cl0_Tb0 (Chave, [Coluna0],
[Coluna1], Data, Hora) VALUES (ChavePrimaria,
'01/01/2001', 2.34, '21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb5_Ln1_Cl0_Tb0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Tb5_Ln1_Cl0_Tb0 (Chave, [Coluna0],
[Coluna1], Data, Hora) VALUES (ChavePrimaria,
'02/04/2001', 3.56, '21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg6;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg6 (Chave, texto, data, hora) VALUES
(ChavePrimaria, 'Exemplo de lista simples: ',
'21/8/2001', '15:33:20');
```

```
SELECT MAX(Chave) AS UltimaChave FROM
Doc_ListaSimples7;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_ListaSimples7 (Chave, Linha, data,
hora) VALUES (ChavePrimaria, 'Primeira linha ',
'21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_ListaSimples7;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_ListaSimples7 (Chave, Linha, data,
hora) VALUES (ChavePrimaria, 'Segunda linha ',
'21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_ListaSimples7;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_ListaSimples7 (Chave, Linha, data,
hora) VALUES (ChavePrimaria, 'Terceira linha ',
'21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg8;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg8 (Chave, texto, data, hora) VALUES
(ChavePrimaria, 'Exemplo de lista de definição: ',
'21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_LD9 (Chave, Definicao, Descricao, Data,
Hora) VALUES (ChavePrimaria, 'Termo 1 ', 'Este termo é
simples ', '21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_LD9 (Chave, Definicao, Descricao, Data,
Hora) VALUES (ChavePrimaria, 'Termo 2 ', 'Possui corpo.
Tabelas começam com Doc_LD9', '21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9_Pg0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_LD9_Pg0 (Chave, texto, data, hora)
VALUES (ChavePrimaria, 'Este termo tem um corpo ',
'21/8/2001', '15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9_Ct1;

PARAMETERS ChavePrimaria INTEGER;
```

```
INSERT INTO Doc_LD9_Ct1 (Chave, texto, data, hora)
VALUES (ChavePrimaria, 'Uma citação ', '21/8/2001',
'15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9_Dv2;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_LD9_Dv2 (Chave, texto, data, hora)
VALUES (ChavePrimaria, 'Uma divisão ', '21/8/2001',
'15:33:20');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg0 (Chave, texto, data, hora) VALUES
(ChavePrimaria, 'Este é um parágrafo de texto normal. O
próximo contém um corpo interno (versão 2): ',
'20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg1 (Chave, data, hora) VALUES
(ChavePrimaria, '20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1_Corpo;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Pg1_Corpo (Chave, texto, data, hora,
ChaveEstrangeira) VALUES (ChavePrimaria, 'Uma citação
(v2). ', '20/8/2001', '16:38:46', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1_Corpo;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Pg1_Corpo (Chave, texto, data, hora,
ChaveEstrangeira) VALUES (ChavePrimaria, 'Uma linha de
código (v2) ', '20/8/2001', '16:38:46',
ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1_Corpo;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg1;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Pg1_Corpo (Chave, texto, data, hora,
ChaveEstrangeira) VALUES (ChavePrimaria, 'Outra citação
(v2). ', '20/8/2001', '16:38:46', ChaveEstrangeira);
```

```
SELECT MAX(Chave) AS UltimaChave FROM Doc_Dv2;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Dv2 (Chave, data, hora) VALUES
(ChavePrimaria, '20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Dv2_Ct0;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Dv2;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Dv2_Ct0 (Chave, data,
hora,ChaveEstrangeira) VALUES (ChavePrimaria,
'20/8/2001', '16:38:46', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Dv2_Ct0_Corpo;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Dv2_Ct0;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Dv2_Ct0_Corpo (Chave, texto, data,
hora, ChaveEstrangeira) VALUES (ChavePrimaria, 'Uma
saída dentro de uma citação, que está inserida em uma
outra hierarquia: uma divisão ', '20/8/2001',
'16:38:46', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Tb3_Cmp (Chave, Data, Hora) VALUES
(ChavePrimaria, '20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt0;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Tb3_Cmp_TabInt0 (Chave, [Coluna0],
Data, Hora, ChaveEstrangeira) VALUES (ChavePrimaria,
'Célula 1 (v2)', '20/8/2001', '16:38:46',
ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt0;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;
```

```
INSERT INTO Doc_Tb3_Cmp_TabInt0 (Chave, [Coluna0],
Data, Hora, ChaveEstrangeira) VALUES (ChavePrimaria,
'Célula 2 (v2)', '20/8/2001', '16:38:46',
ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt0;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Tb3_Cmp_TabInt0 (Chave, [Coluna0],
Data, Hora, ChaveEstrangeira) VALUES (ChavePrimaria,
'Célula 3 (v2)', '20/8/2001', '16:38:46',
ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt1;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Tb3_Cmp_TabInt1 (Chave, [Grupo 2 1],
[Grupo 2 2], Data, Hora, ChaveEstrangeira) VALUES
(ChavePrimaria, '01/08/2001', 51, '20/8/2001',
'16:38:46', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt1;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Tb3_Cmp_TabInt1 (Chave, [Grupo 2 1],
[Grupo 2 2], Data, Hora, ChaveEstrangeira) VALUES
(ChavePrimaria, '04/09/2001', 97, '20/8/2001',
'16:38:46', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb3_Cmp_TabInt1;

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb3_Cmp;

PARAMETERS ChavePrimaria INTEGER, ChaveEstrangeira
INTEGER;

INSERT INTO Doc_Tb3_Cmp_TabInt1 (Chave, [Grupo 2 1],
[Grupo 2 2], Data, Hora, ChaveEstrangeira) VALUES
(ChavePrimaria, '05/05/2001', 34, '20/8/2001',
'16:38:46', ChaveEstrangeira);

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg4;

PARAMETERS ChavePrimaria INTEGER;
```

```
INSERT INTO Doc_Pg4 (Chave, texto, data, hora) VALUES
(ChavePrimaria, 'A próxima tabela tem uma tabela
interna: (v2) ', '20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Tb5;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Tb5 (Chave, [Coluna 1], Data, Hora)
VALUES (ChavePrimaria, 'Possui corpo. Tabelas começam
com Doc_Tb5_Ln1_Cl0', '20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb5_Ln1_Cl0_Tb0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Tb5_Ln1_Cl0_Tb0 (Chave, [Coluna0],
[Coluna1], Data, Hora) VALUES (ChavePrimaria,
'01/08/2001', 2.34, '20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb5_Ln1_Cl0_Tb0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Tb5_Ln1_Cl0_Tb0 (Chave, [Coluna0],
[Coluna1], Data, Hora) VALUES (ChavePrimaria,
'03/03/2001', 2, '20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_Tb5_Ln1_Cl0_Tb0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Tb5_Ln1_Cl0_Tb0 (Chave, [Coluna0],
[Coluna1], Data, Hora) VALUES (ChavePrimaria,
'02/09/2001', 3.56, '20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg6;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg6 (Chave, texto, data, hora) VALUES
(ChavePrimaria, 'Exemplo de lista simples: (v2) ',
'20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_ListaSimples7;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_ListaSimples7 (Chave, Linha, data,
hora) VALUES (ChavePrimaria, 'Primeira linha (v2) ',
'20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_ListaSimples7;

PARAMETERS ChavePrimaria INTEGER;
```

```
INSERT INTO Doc_ListaSimples7 (Chave, Linha, data,
hora) VALUES (ChavePrimaria, 'Segunda linha (v2) ',
'20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_ListaSimples7;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_ListaSimples7 (Chave, Linha, data,
hora) VALUES (ChavePrimaria, 'Terceira linha (v2) ',
'20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM
Doc_ListaSimples7;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_ListaSimples7 (Chave, Linha, data,
hora) VALUES (ChavePrimaria, 'Quarta linha (v2) ',
'20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_Pg8;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_Pg8 (Chave, texto, data, hora) VALUES
(ChavePrimaria, 'Exemplo de lista de definição: (v2) ',
'20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_LD9 (Chave, Definicao, Descricao, Data,
Hora) VALUES (ChavePrimaria, 'Termo 1 (v2) ', 'Este
termo é simples (v2) ', '20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_LD9 (Chave, Definicao, Descricao, Data,
Hora) VALUES (ChavePrimaria, 'Termo 2 (v2) ', 'Possui
corpo. Tabelas começam com Doc_LD9', '20/8/2001',
'16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9_Pg0;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_LD9_Pg0 (Chave, texto, data, hora)
VALUES (ChavePrimaria, 'Este termo tem um corpo (v2) ',
'20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9_Ct1;

PARAMETERS ChavePrimaria INTEGER;
```

```
INSERT INTO Doc_LD9_Ct1 (Chave, texto, data, hora)
VALUES (ChavePrimaria, 'Uma citação (v2) ',
'20/8/2001', '16:38:46');

SELECT MAX(Chave) AS UltimaChave FROM Doc_LD9_Dv2;

PARAMETERS ChavePrimaria INTEGER;

INSERT INTO Doc_LD9_Dv2 (Chave, texto, data, hora)
VALUES (ChavePrimaria, 'Uma divisão (v2) ',
'20/8/2001', '16:38:46');
```

ANEXO 4 – CONTÉUDO DAS RELAÇÕES GERADAS PELO PROCESSAMENTO DE DUAS VERSÕES DE UM DOCUMENTO

Doc_Pg0

Chave	Texto	Data	Hora
1	Este é um parágrafo de texto normal. O próximo contém um corpo interno:	21/08/01	15:33:20
2	Este é um parágrafo de texto normal. O próximo contém um corpo interno (versão 2):	20/08/01	16:38:46

Doc_Pg1

Chave	Data	Hora
1	21/08/01	15:33:20
2	20/08/01	16:38:46

Doc_Pg1_Corpo

Chave	Texto	Data	Hora	ChaveEstrangeira
1	Uma citação.	21/08/01	15:33:20	1
2	Uma linha de código	21/08/01	15:33:20	1
3	Outra citação.	21/08/01	15:33:20	1
4	Uma citação (v2).	20/08/01	16:38:46	2
5	Uma linha de código (v2)	20/08/01	16:38:46	2
6	Outra citação (v2).	20/08/01	16:38:46	2

Doc_Dv2

Chave	Data	Hora
1	21/08/01	15:33:20
2	20/08/01	16:38:46

Doc_Dv2_Ct0

Chave	Data	Hora	ChaveEstrangeira
1	21/08/01	15:33:20	1
2	20/08/01	16:38:46	2

Doc_Dv2_Ct0_Corpo

Chave	Texto	Data	Hora	ChaveEstrangeira
1	Uma saída dentro de uma citação, que está inserida em uma outra hierarquia: uma divisão.	21/08/01	15:33:20	1
2	Uma saída dentro de uma citação, que está inserida em uma outra hierarquia: uma divisão	20/08/01	16:38:46	2

Doc_Tb3_Cmp

Chave	Data	Hora
1	21/08/01	15:33:20
2	20/08/01	16:38:46

Doc_Tb3_Cmp_TabInt0

Chave	Coluna0	Data	Hora	ChaveEstrang
1	Célula 1	21/08/01	15:33:20	1
2	Célula 2	21/08/01	15:33:20	1
3	Célula 1 (v2)	20/08/01	16:38:46	2
4	Célula 2 (v2)	20/08/01	16:38:46	2
5	Célula 3 (v2)	20/08/01	16:38:46	2

Doc_Tb3_Cmp_TabInt1

Chave	Grupo 2 1	Grupo 2 2	Data	Hora	ChaveEstrang
1	01/01/01	5	21/08/01	15:33:20	1
2	04/02/01	9	21/08/01	15:33:20	1
3	01/08/01	51	20/08/01	16:38:46	2
4	04/09/01	97	20/08/01	16:38:46	2
5	05/05/01	34	20/08/01	16:38:46	2

Doc_Pg4

Chave	Texto	Data	Hora
1	A próxima tabela tem uma tabela interna:	21/08/01	15:33:20
2	A próxima tabela tem uma tabela interna: (v2)	20/08/01	16:38:46

Doc_Tb5

Chave	Coluna 1	Data	Hora
1	Possui corpo. Tabelas começam com Doc_Tb5_Ln1_Cl0	21/08/01	15:33:20
2	Possui corpo. Tabelas começam com Doc_Tb5_Ln1_Cl0	20/08/01	16:38:46

Doc_Tb5_Ln1_Cl0_Tb0

Chave	Coluna0	Coluna1	Data	Hora
1	01/01/01	2,34	21/08/01	15:33:20
2	02/04/01	3,56	21/08/01	15:33:20
3	01/08/01	2,34	20/08/01	16:38:46
4	03/03/01	2	20/08/01	16:38:46
5	02/09/01	3,56	20/08/01	16:38:46

Doc_Pg6

Chave	Texto	Data	Hora
1	Exemplo de lista simples:	21/08/01	15:33:20
2	Exemplo de lista simples: (v2)	20/08/01	16:38:46

Doc_ListaSimples7

Chave	Linha	Data	Hora
1	Primeira linha	21/08/01	15:33:20
2	Segunda linha	21/08/01	15:33:20
3	Terceira linha	21/08/01	15:33:20
4	Primeira linha (v2)	20/08/01	16:38:46
5	Segunda linha (v2)	20/08/01	16:38:46
6	Terceira linha (v2)	20/08/01	16:38:46
7	Quarta linha (v2)	20/08/01	16:38:46

Doc_Pg8

Chave	Texto	Data	Hora
1	Exemplo de lista de definição:	21/08/01	15:33:20
2	Exemplo de lista de definição: (v2)	20/08/01	16:38:46

Doc_LD9

Chave	Definicao	Descricao	Data	Hora
1	Termo 1	Este termo é simples	21/08/01	15:33:20
2	Termo 2	Possui corpo. Tabelas começam com Doc_LD9	21/08/01	15:33:20
3	Termo 1 (v2)	Este termo é simples (v2)	20/08/01	16:38:46
4	Termo 2 (v2)	Possui corpo. Tabelas começam com Doc_LD9	20/08/01	16:38:46

Doc_LD9_Ct1

Chave	Texto	Data	Hora
1	Uma citação	21/08/01	15:33:20
2	Uma citação (v2)	20/08/01	16:38:46

Doc_LD9_Dv2

Chave	Texto	Data	Hora
1	Uma divisão	21/08/01	15:33:20
2	Uma divisão (v2)	20/08/01	16:38:46

Doc_LD9_Pg0

Chave	Texto	Data	Hora
1	Este termo tem um corpo	21/08/01	15:33:20
2	Este termo tem um corpo (v2)	20/08/01	16:38:46