UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DESARROLLO DE UN CONSTRUCTOR AUTOMÁTICO
DE BASES DE DATOS RELACIONALES
A PARTIR DE ESQUEMAS XML

T E S I S

QUE PARA OBTENER EL TÍTULO DE:
INGENIERO EN COMPUTACIÓN

P R E S E N T A:
AXEL GERARDO GARDUÑO SANDOVAL

DIRECTOR DE TESIS:
DR. ALFONSO MEDINA URREA

MÉXICO, D.F. 2010

Índice

1. Introducción

Hoy en día existen muchas investigaciones relacionadas con el diseño y compilación de corpus lingüísticos electrónicos. Los corpus electrónicos son colecciones de documentos textuales en forma electrónica que constituyen una muestra del uso del lenguaje en algún ámbito general o específico. Es decir, conjuntos de documentos representativos de la forma en que se habla y se escribe algún idioma como el español o algún lenguaje de especialidad como el utilizado en las ingenierías.

Las investigaciones relacionadas con los corpus se hacen desde diversas perspectivas, ya que tienen muchas aplicaciones. Por ejemplo, la lingüística se ha beneficiado enormemente de la disponibilidad de estos recursos y las herramientas computacionales para analizarlos. Otro ejemplo importante está constituido por las investigaciones en computación para el procesamiento de grandes cantidades de textos. Esto último está muy en boga por el auge que vive Internet y los diversos desarrollos de minería de textos para descubrir información importante en grandes cantidades de textos.

En el Instituto de Ingeniería de la UNAM se lleva a cabo investigación para desarrollar estos recursos y hacer investigación básica y aplicada en lingüística y computación para, entre otras cosas, desarrollar tecnologías del lenguaje, tales

como diccionarios electrónicos, analizadores morfológicos y sintácticos, sintetizadores de voz, buscadores de contextos definitorios, etc.

En ese marco y mediante el financiamiento DGAPA PAPIIT de los proyectos IN400905 “Constitución del Corpus Histórico del Español de México” e IN402008 “Glutinometría y variación dialectal”, el Grupo de Ingeniería Lingüística desarrolla varios corpus electrónicos. El que atañe a esta tesis es específicamente el Corpus Histórico del Español en México, que consiste en diversos documentos escritos en los siglos XVI, XVII, XVIII y XIX, tanto en la Nueva España como en el México Independiente.

Los documentos de este corpus están codificados en XML (***Extensible Markup Language***, o en español, lenguaje de etiquetado extensible). Típicamente, existe una base de datos bibliográfica para concentrar en un lugar la información sobre autores, títulos de documentos, etc. de estos documentos. El presente trabajo contribuye a automatizar la construcción de esta base de datos. Específicamente, este trabajo consiste en el desarrollo de un constructor de bases de datos bibliográficas que analiza el esquema XML que valida los documentos del corpus para conocer la estructura de la información bibliográfica que servirá para dicha base.

Para llevar a cabo este trabajo, a continuación se enuncian los objetivos de esta tesis, la definición del problema a resolver y la metodología que se utilizará para llevar a cabo estas tareas.

Objetivo de la tesis

Desarrollar un constructor automático de bases de datos relacionales a partir de esquemas XML para ser pobladas con datos en documentos XML

Definición del problema

Se suele introducir manualmente información en bases de datos sujeta a errores de varios niveles. La ventaja de los documentos XML es que tienen la información pertinente ya codificada. Los esquemas XML, además de validar la estructura de esos documentos, representan la forma de la base de datos. Queremos construir dicha base a partir de estos esquemas y llenarla automáticamente con la información codificada en los documentos, evitando errores y automatizando el proceso.

Metodología

1. Se hará un modelado UML del desarrollo.
2. Se formulará un algoritmo que se irá mejorando iterativamente según se vayan incorporando las características del esquema.
3. Se desarrollará, instalará y comprobará el funcionamiento adecuado del constructor.

En los siguientes capítulos se expondrá la información básica sobre XML en general como lenguaje de codificación de datos en documentos (capítulo 2), la aplicación de la tecnología XML en el ***Corpus Histórico del Español en México*** (capítulo 3), además se hablará de SQL como lenguaje de consultas de bases de datos (capítulo 4). En el capítulo 5, se presenta la metodología seguida para el desarrollo del presente trabajo. En los siguientes capítulos (6 y 7), se describen

los comandos SQL para la construcción y población de la base de datos. En el capítulo 8 se presentan las conclusiones y en el apéndice al final se incluyen los diagramas UML generados y la bibliografía consultada.

2. La tecnología XML

En este capítulo[1], se examinan las generalidades del lenguaje de codificación de documentos XML (***Extensible Markup Language***, o en español, lenguaje de etiquetado extensible). En esencia, trata de qué se puede hacer y qué no se puede hacer con documentos XML. Además, se mencionan cuáles son las principales ventajas y diferencias que hay al utilizar este tipo de etiquetado de documentos, comparados con documentos HTML. Por último, se enumeran las características que deben cumplir los documentos de texto para que éste sea un documento bien formado, y posteriormente validarlo usando una DTD o un esquema XML.

Generalidades

Qué no es XML

Antes de empezar a describir lo que son los documentos XML, y las características que deben tener los documentos de texto para ser considerados como XML, voy a hacer algunas precisiones, para explicar lo que es y no es un documento XML y lo que se puede y no se puede hacer con ellos.

---

[1] La información presentada aquí se obtuvo de revisar las siguientes fuentes: el libro de MacDonald, Matthew (***Office 2003 XML for Power Users***, Apress, 2004) y las siguientes páginas:

- http://es.wikipedia.org/wiki/XML
- http://es.wikipedia.org/wiki/Validacion_xml
- http://manuales.dgsca.unam.mx/xml/Qu%E9%20es%20DTD.htm
- http://www.sidar.org/recur/desdi/traduc/es/xml/xml10p/xml10p.htm
- http://www.w3.org/TR/xml11/
- http://es.geocities.com/alba1509/Fase3/teoria.html
- http://www.scribd.com/doc/6974950/XML
- http://www.webtaller.com/construccion/lenguajes/xml/lecciones/xml-10-puntos.php

Primero, XML no es un lenguaje de programación. Los documentos XML no son programas y no pueden “correr” o “ejecutar”. Segundo, aunque los documentos XML tienen una estructura arbórea, y esa característica se utilizó en esta tesis para que a partir de ella se pueda crear una base de datos relacional, XML en un sentido estricto sólo es una forma de delimitar piezas de datos, no es una base de datos.

Por otra parte, es importante mencionar que aunque XML y HTML tienen ciertas semejanzas, empezando por los caracteres ML (***Markup Language***) del nombre, son lenguajes de codificación diferentes. Tienen en común que utilizan etiquetas y atributos, (<etiqueta atributo="valor" …> delimitada por picoparéntesis < >). La diferencia consiste en que HTML tiene un conjunto (relativamente cerrado) de etiquetas con estructuras especificadas por quienes diseñaron el lenguaje, dicho de otra forma, se especifica lo que cada etiqueta y atributo significan, también nos indican la apariencia que deben tener los documentos en los navegadores (el texto, su tamaño, tipo de letra, color, su ubicación, y hasta el fondo que debe tener el texto). En cambio, XML utiliza las etiquetas para delimitar y estructurar piezas de datos según las especificaciones y necesidades del usuario y deja la interpretación de los datos, completamente, a la aplicación que los lee. Por ejemplo, si en un documento HTML se observa una etiqueta "<p>", esto quiere decir que se trata de un párrafo, en cambio si encontramos la etiqueta "<p>", en un documento XML, no necesariamente se trata de un párrafo; es decir, dependiendo de cómo se quiera usar, puede tratarse de un precio, un parámetro,

una persona, u otra cosa. Además, si quisiéramos marcar un párrafo lo podríamos hacer también llamándolo <parrafo>, debido a que en los documentos XML cada usuario tiene la libertad de especificar el nombre de las etiquetas según convenga.

En pocas palabras se puede decir que XML es un lenguaje similar a HTML pero su función principal es describir datos y no mostrarlos como es el caso de HTML. Conviene hacer notar que XML es un formato que permite la lectura de datos a través de diferentes aplicaciones y no se creo para sustituir a HTML, debido a las diferencias antes mencionadas.

Una vez que ya se examinaron las generalidades de los documentos XML y también se mostraron algunas diferencias que existen entre un documento HTML y uno XML, empezaré a describir las características que deben cumplir los documentos de texto para que puedan ser llamados documentos XML.

XML

La especificación o documento que define cómo diseñar y aplicar las etiquetas y los atributos en XML es ***XML 1.0***. Alrededor de esta especificación se crearon diferentes módulos opcionales que tienen colecciones de etiquetas y atributos, o modelos para especificar tareas que pueden tanto presentar la información u otra cosa. Por ejemplo:

- ***Xlink,***
- ***CSS,***

- ***XSL ,***
- ***XSLT,***
- ***DOM ,***
- ***XML Namespaces,***
- ***Esquemas XML (XML Schemas),***
- entre otros.

Respecto a los documentos XML, éstos están formados por un prólogo y por el cuerpo del documento. El prólogo es una etiqueta donde se especifica la versión XML, el tipo de documento y otras cosas (`<?xml version="1.0" … ?>`). Esta primera etiqueta contiene la información de la versión de manera obligatoria, (hay que mencionar que aunque la versión más utilizada es la 1.0, ya está disponible la versión 1.1 de XML[2]), dentro de la misma etiqueta también se incluye opcionalmente la codificación de caracteres utilizada (***encoding***); ésta hace referencia al modo en que se representan internamente los caracteres, normalmente UTF-8 ó UTF-16 ó Unicode. Y por último, también en esta primera etiqueta, se puede incluir la declaración independiente (standalone) que indica al procesador XML si un documento es independiente (standalone="yes") o se basa en información de fuentes externas, es decir, si depende de declaraciones de marca externas como una DTD externa (standalone="no"), esta es la opción por defecto.

A continuación se muestra un ejemplo de la primera etiqueta:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
```

[2] http://www.w3.org/TR/xml11/

Después de esta etiqueta sigue la declaración de tipo de documento. Ésta enlaza el documento con su DTD (definición de tipo de documento) o esquema XML. Opcionalmente, el DTD puede estar incluido en la propia declaración o ambas cosas al mismo tiempo.

Luego sigue el cuerpo del documento. La primera característica indispensable que debe tener para que pueda considerarse documento XML es que debe contener un y sólo un elemento raíz para cada documento; esto es, un solo elemento en el que todos los demás elementos estén contenidos dentro de éste. Estos elementos contenidos se encuentran anidados y correctamente cerrados, es decir, se basa en una estructura jerárquica y su función es la misma que el elemento raíz de un documento HTML `<HTML>Contenido</HTML>.`

Todas las etiquetas (que representan elementos o entidades) utilizadas en el documento se declaran en una DTD interna o externa, o en un esquema XML. Todos los elementos, atributos y entidades que se utilicen deben escribirse con una sintaxis correcta, según esta DTD o esquema XML. Algunas de las características más importantes que debe tener un documento XML se mencionan a continuación.

Todos los elementos deben estar delimitados por una etiqueta inicial y otra final con el mismo nombre en el siguiente formato:

```
<etiqueta></etiqueta >
```

Los documentos siguen una estructura estrictamente jerárquica respecto a las etiquetas que delimitan sus elementos. Una etiqueta debe estar correctamente incluida dentro de otra. Esto quiere decir que las etiquetas deben anidarse correctamente. Obviamente, todos los elementos deben estar cerrados de la manera apropiada.

```
<documento>
          <titutlo>Historia</titulo>
          <fechaPublicacion>1821</fechaPublicacion>
</documento >
```

Los atributos de una etiqueta en XML deben estar contenidos dentro de esta y los valores de dichos atributos deben ir entre comillas dobles. Los elementos vacíos deben terminar con '/' (autocierre), <ejemplo/>, o añadiendo una etiqueta de fin, <ejemplo></ejemplo>, y no puede haber etiquetas no cerradas. Los siguientes ejemplos contienen dos atributos ***tipo*** y ***pags,*** en el primero existe una etiqueta de cierre, y la segunda con autocierre, ambos son correctos:

```
<otro tipo="acta" pgs="1"></otro>
<otro tipo="acta" pgs="1"/>
```

XML es sensible a mayúsculas y minúsculas y los nombres de las etiquetas pueden ser alfanuméricos. También hay que hacer notar que existe un conjunto de caracteres que se pudieran ser considerados como espacios en blanco (espacios, tabulaciones, saltos de línea, entre otros) que en los documentos XML, cada uno de estos se consideran caracteres diferentes en el marcado XML.

Las etiquetas, y todos los elementos que se encuentran dentro de los picoparéntesis, son partes del documento que el procesador XML puede entender. El resto del documento, es la información que se espera sea leída por los usuarios.

A continuación se presenta un ejemplo sencillo, en dónde se cumple con las condiciones que se mencionaron anteriormente:

```
<?xml version="1.0" encoding=" UTF-8" ?>
      <referencias>
            <referencia id="chem0">
                  <lugar>Apatzingan</lugar>
                  <otro tipo="decreto" pgs="1"/>
                  <corta><i>Decreto constitucional para la libertad de la
          América Mexicana</i></corta>
            </referencia>

            <referencia id="chem1">
                  <otro tipo="acta" pgs="1"/>
                  <corta><i>Acta de independencia del Imperio
          Mexicano</i></corta>
                  </referencia>
      </referencias>
```

Si un documento de texto cumple con las condiciones anteriores que son las especificaciones de marcado XML, entonces se dice que dicho documento esta “bien formado” (***well formed***). Al cumplir con esta característica dichos documentos pueden ser analizados por un parser, para verificar que siga la sintaxis XML. Hay que mencionar que no son sinónimos que un documento sea “bien formado” y que sea válido.

La sintaxis XML establece los requisitos mínimos que debe cumplir un documento XML. Si se quiere tener un mayor control del contenido de los documentos y aprovechar dicha información de una mejor manera, es necesario establecer un conjunto de definiciones más adecuadas a nuestras necesidades, este proceso es llamado validación de los documentos XML.

Validación XML

La validación es la parte más importante dentro de esta exposición, porque determina si un documento creado se ajusta a las restricciones descritas en el esquema utilizado para su elaboración. Es cierto que se pueden utilizar documentos que no se encuentren asociados a ningún esquema y por lo tanto no tendríamos necesidad de validarlos. Sin embargo, en la mayoría de los casos conviene que estén validados para aprovechar al máximo las ventajas de los documentos XML.

La validación XML es la comprobación de que un documento en lenguaje XML está bien formado y se ajusta a una estructura definida. Un documento bien formado sigue las reglas de sintaxis de XML, pero un documento válido además de cumplir con lo anterior respeta las normas establecidas por su DTD o esquema XML utilizado. Controlar el diseño de documentos a través de esquemas aumenta su nivel de fiabilidad, consistencia y precisión, logrando con esto un mejor manejo entre diferentes aplicaciones y usuarios. Cuando creamos documentos XML válidos logramos que estos se ajusten de una mejor manera, a las necesidades que nosotros requerimos.

Métodos de validación

Aunque ya se mencionó la importancia de validación o validar los documentos XML, conviene mencionar que existen varios métodos para validar los documentos XML. Los métodos más usados son la ***DTD*** de XML versión 1.0, el ***XML Schema*** de W3C, aunque no son los únicos (ver por ejemplo: RELAX NG, Schematron).

A continuación se explican de una forma más detallada los DTD y los esquemas XML que se utilizan para validad un documento XML, describiendo algunas de sus principales características y ventajas.

DTD

La DTD (document type definition) es el formato de esquema nativo (y el más antiguo) para validar documentos XML, heredado de SGML. Dichos esquemas utilizan una sintaxis diferente a la de XML para definir la estructura o modelo de contenido de un documento XML válido:

- Define todos los elementos.
- Define todas las relaciones entre los distintos elementos.
- Proporciona toda información adicional que pueda ser incluida en el documento (atributos, entidades, notaciones).
- Aporta comentarios e instrucciones para su procesamiento y para la representación de los formatos de datos.

Es el método más antiguo usado para validar. Las DTDs pueden estar asociadas a un documento XML de manera interna o externa, o de ambas

maneras a la vez; esto es, parte de una puede estar contenida dentro del documento XML, mientras que la otra puede estar en un archivo de texto separado.

Algunas de las principales desventajas de este tipo de esquema de validación, es que el DTD es poco flexible para definir elementos con contenido mixto, es decir, que incluyan otros elementos además de texto. Además, se complica indicar a qué tipo de dato (número, fecha, moneda) ha de corresponder un atributo o el texto de un elemento.

Ejemplo de un DTD:

```
<!ELEMENT lista_de_personas (persona*)>
<!ELEMENT persona (nombre, fechanacimiento?, sexo?, numeroseguridadsocial?)>
<!ELEMENT nombre (#PCDATA) >
<!ELEMENT fechanacimiento (#PCDATA) >
<!ELEMENT sexo (#PCDATA) >
<!ELEMENT numeroseguridadsocial (#PCDATA)>
```

Esquema XML (*XML Schema*)

Ya que se explicó brevemente cuales son las característica de un DTD, sin embargo hay que destacar que existe un formato que nos facilita la elaboración de esquemas, y nos permite especificar además de estructura, también anidación, restricciones y tipos de dato complejos con base en tipos de dato más simples, cosa que no se puede realizar utilizando una DTD. El esquema XML se creó para solucionar algunas limitaciones que se presentaban en los DTD, especialmente en lo referente a lo complicado que pudiera ser definir tipos de datos que no sean de

texto puro y la falta de jerarquización en las DTD. Tanto los DTD como el esquema XSD, están descritas por el W3C (Consorcio World Wide Web) que es un consorcio donde se desarrollan estándares de WEB. El esquema XML, también llamado XSD (***XML Schema Definition***), es un lenguaje de representación más completo y más poderoso que el de una DTD y debido a que se pueden declarar un número mayor de tipos de datos, además de que tienen una estructura jerárquica lo que facilita la creación de este tipo de documentos; asimismo utiliza una sintaxis parecida a la de XML, lo que le permite especificar de forma más detallada un sistema, gracias a que cuenta con un extenso de tipos de datos y se pueden crear los propios. A diferencia de las DTDs, soporta la extensión del documento sin mayor problema.

Una de las desventajas es que a la hora de validar, no todos los parser contienen la información necesaria para poder validar un documento que haya utilizado un esquema XSD, al contrario de los DTD que están contenidos en casi todos los programas que se utilizan para validar un documento XML, además de que debido a sus características las definiciones pueden ser complejas, lo que provoca mayor gasto de recursos al momento de validar, pero es una desventaja menor comparada con la gran cantidad de cosas que nos permite realizar.

Hay que mencionar que el uso de las DTD y esquemas XML, son los más utilizados actualmente, pero no son los únicos y existen otros “esquemas descriptivos” que nos facilitan ciertas tareas especificas, aunque con los esquemas XSD se puede definir casi la totalidad de las necesidades que

deseemos. En algunas ocasiones resulta muy complicado ó redundante la definición de algunos tipos de datos, para estos casos especiales se crean otros esquemas que nos dan atajos para este tipo de necesidades especificas; hay algunos que son reconocidos por la W3C, y hay otros que a pesar de su amplia utilización no se consideran todavía dentro de las recomendaciones de la W3C, algunos de ellos son: RELAX NG, Schematron, Namespace Routing Language (NRL), Document Schema Definition Languages (DSDL), Document Definition Markup Language (DDML), Document Structure Description (DSD), SGML, Schema for Object-Oriented XML (SOX).

Ejemplo esquema XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3c.org/2001/XMLSchema">
     <xsd:element name="Libro">
         <xsd:complexType>
            <xsd:sequence>
                 <xsd:element name="Título" type="xsd:string"/>
                 <xsd:element       name="Autores"       type="xsd:string"
maxOccurs="10"/>
                 <xsd:element name="Editorial" type="xsd:string"/>
            </xsd:sequence>
            <xsd:attribute name="precio" type="xsd:double"/>
        </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Ventajas de los esquemas XML frente a los DTDs

Algunas de las ventajas que tienen los esquemas XSD, mas importantes y que hacen que sea mas robusto comparado con un DTD son las siguientes:

- Su sintaxis está basada en la de XML, al contrario que los DTDs, en la que su sintaxis puede resultar algo confusa, especialmente si no se ha manejado este tipo de esquema con anterioridad.

- Se puede manejar en términos generales como cualquier otro documento XML, ya que este tipo de esquemas tienen una estructura jerárquica.
- Permiten especificar los tipos de datos, ya que no están limitados a los que se definan por el propio esquema, sino que pueden existir tipos de datos complejos creados por la persona que usa el esquema XML.
- Son extensibles.

Ventajas de la tecnología XML

Las ventajas que se tiene al utilizar documentos XML, pueden enumerarse en una gran cantidad, sin embargo existen algunas que son de mayor importancia. Comenzando por el hecho de que el lenguaje es extensible y es definido por la persona que lo utiliza, lo cual nos permite generar etiquetas entendibles por el usuario que las crea adaptándose a sus necesidades, y una vez diseñado el lenguaje y puesto a funcionar, es posible aumentarlo usando nuevas etiquetas para describir partes de texto de una mejor manera o para describir algún tipo de información que no había sido considerada o que no existía, de manera que se puedan usar las definiciones sencillas, a la vez que se usen definiciones más detalladas, y ambas, tanto nuevas como antiguas convivan sin mayor problema.

Otra de las ventajas de utilizar este tipo de documentos, es que debido a la estructura arbórea propia de los documentos XML, es sencillo de entender de lo que tratan los documentos aún sin conocerlos detalladamente, ya que tales documentos presentan un orden. Añadiendo que actualmente existe una gran cantidad de aplicaciones que nos permiten validar los documentos de una manera rápida y sencilla.

Por último hay que agregar que existen diferentes opciones para generar formatos de texto a partir de la información previamente codificada en XML, creando a partir de un documento XML, diferentes formatos de texto según lo necesitemos.

Aquí se termina el capítulo que abarca de forma general lo que son los documentos XML, y sus características que sirvieron de base para realizar el análisis, tanto de los documentos XML como de los esquemas XSD, para posteriormente crear el algoritmo que me permitiera desarrollar el constructor automático de la base de datos. En el siguiente capítulo se abordará lo referente a la aplicación de la tecnología XML en el ***Corpus Histórico del Español en México***, mostrando de una forma general un documento y cómo está codificado.

3. XML en el *Corpus Histórico del Español en México*

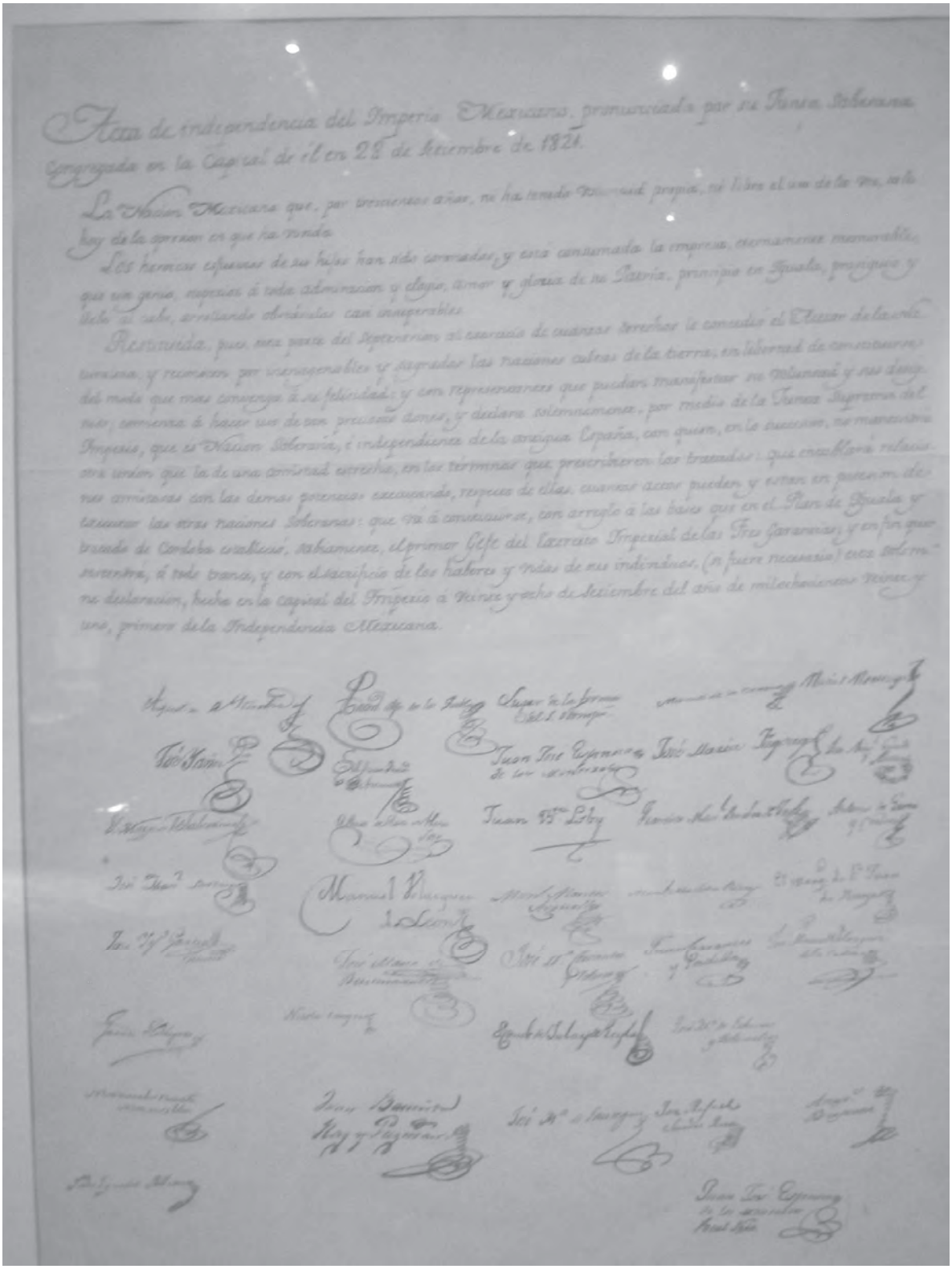
En este capítulo se muestran aspectos básicos de la utilización de XML en la codificación del ***Corpus Histórico del Español en México***, al que esta tesis se enfoca. Esto es importante para describir el contexto en el que se desarrolla el constructor automático de la base bibliográfica de dicho corpus. En esencia, se presentará un documento XML, el esquema XML que lo valida y que el constructor analizará para construir la base de datos y la porción del documento XML que servirá para poblar la base de datos creada.

Los documentos XML, como se dijo arriba, están formados de etiquetas con picoparéntesis y se ven como en la figura 3.1, que contiene el Acta de Independencia del Imperio Mexicano (1821) ya codificada. El documento original se muestra en la figura 3.2.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<?xml-stylesheet type="text/xsl" href="estilo.xsl"?>

<documento      xmlns="http://www.ii.unam.mx"
                xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xsi:schemaLocation="http://www.ii.unam.mx esquema.xsd">

    <encabezado id="0" permisos="todos">

        <titulo>Acta de independencia del Imperio Mexicano</titulo>

        <parametros     generoLiterario="prosa" registro="estándar">
            <corpus><CHEM   zonaGeografica="Altiplano Central" areaTematica="derecho"/></corpus>
            <hablante   genero="grupo masculino"/>
        </parametros>

        <institucion>Junta Soberana del Imperio Mexicano</institucion>
        <referencia><otro tipo="acta" pgs="1"/></referencia>
        <ciudadPublicacion>ciudad de México</ciudadPublicacion>
        <fechaPublicacion>1821</fechaPublicacion>
        <imagen origen="internet" ancho="400" alto="532">Independencia-mx-acta.jpg</imagen>

        <responsables>
            <transcriptor nombres="" apellidos="" fecha="noviembre 2007"/>
            <etiquetador nombres="" apellidos="" fecha="noviembre 2007"/>
            <revisor nombres="" apellidos="" fecha="noviembre 2007"/>
        </responsables>

    </encabezado>

<cuerpo tipoFuente="Espinosa">
<bastardillas>
<seccion>
<tituloSecc>
    <g>Acta</g><e/>
    <g>de</g><e/>
    <g>independencia</g><e/>
    <g>del</g><e/>
    <g>Imperio</g><e/>
    <g>Mexicano</g><r c="Fc">,</r><e/>
    <g>pronunciada</g><e/>
    <g>por</g><e/>
```

Figura 3.1. Acta de Independencia codificada en XML

Como se observa en la figura 3.1, el documento está formado por dos elementos principales: un encabezado y un cuerpo. Por un lado, el encabezado contiene generalidades del documento, como sus características de clasificación en el corpus (su género literario, registro dialectal, tipo de hablantes, etc.) la información bibliográfica (referencias, lugar y fecha de publicación, etc.) que servirá para poblar la base de datos que generará el constructor de esta tesis y datos sobre los diversos responsables del documento en cuestión. Por el otro, el



Acta de independencia del Imperio Mexicano, pronunciada por su Junta Soberana congregada en la Capital de él en 28 de Setiembre de 1821.

La Nacion Mexicana que, por trescientos años, no ha tenido voluntad propia, ni libre el uso de su voz, hoy sale de la opresion en que ha vivido.

Los heroicos esfuerzos de sus hijos han sido coronados, y está consumada la empresa, eternamente memorable, que un genio superior á toda admiracion y elogio, amor y gloria de su Patria, principió en Iguala, prosiguió y llevó al cabo, arrollando obstáculos casi insuperables.

Restituida, pues, esta parte del Septentrion al exercicio de cuantos derechos le concedió el Criador de la sociedad, y recobrada por irrenunciables y sagrados las naciones cultas de la tierra; en libertad de constituirse del modo que mas convenga á su felicidad; y con representantes que pueden manifestar su voluntad y sus designios; comienza á hacer uso de tan preciosas dotes, y declara solemnemente, por medio de la Junta Suprema del Imperio, que es Nacion Soberana, é independiente de la antigua España, con quien, en lo sucesivo, mantendrá otra union que la de una amistad estrecha, en los términos que prescribieren los tratados: que establecerá relaciones amistosas con las demas potencias executando, respecto de ellas, cuantos actos puedan y estan en posesion de executar las otras naciones Soberanas: que va á constituirse, con arreglo á las bases que fijó en el Plan de Iguala y tratados de Córdoba el citado Sr. Iturbide, primer Gefe del Exército Imperial de las Tres Garantías; y en fin que sostendrá, á toda costa, y con el sacrificio de los haberes y vidas de sus individuos, (si fuere necesario) esta solemne declaracion, hecha en la capital del Imperio á veinte y ocho de Setiembre del año de mil ochocientos veinte y uno, primero de la Independencia Mexicana.

Figura 3.2. Imagen del Acta de Independencia del Imperio Mexicano (1821)

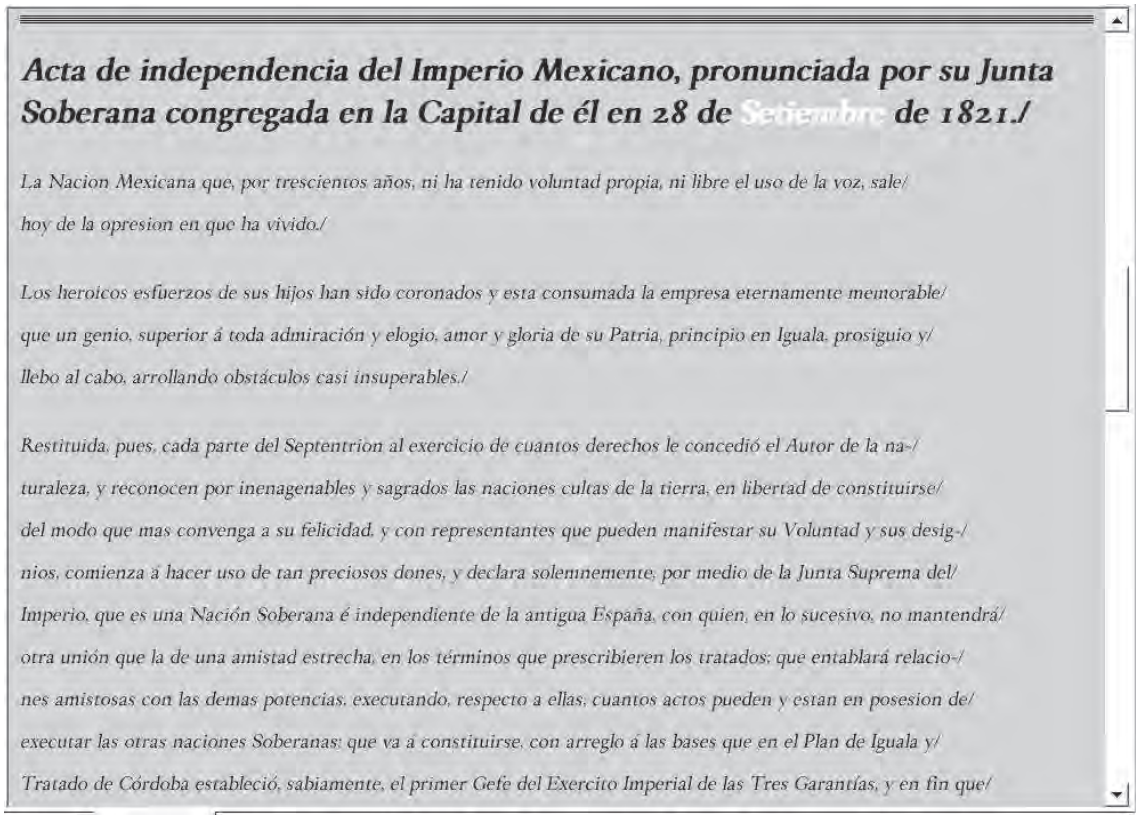
cuerpo contiene el documento mismo, es decir la secuencia específica de “tokens” u ocurrencias de palabras con las particularidades tipográficas del original (tipo de fuente, letras cursivas, bastardillas, versales, negritas, etc.), así como títulos, subtítulos, firmas y errores del original, todo codificado como ya se dijo en XML mediante elementos y atributos diseñados específicamente para este corpus. Todos estos elementos y atributos se validan mediante el esquema XML, parte del cual aparece en la figura 3.3.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

    <xsd:element name="documento">
        <xsd:complexType>
            <xsd:sequence>
                <xsd:element ref="encabezado" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="cuerpo" minOccurs="1" maxOccurs="1"/>
            </xsd:sequence>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="corpus">
        <xsd:complexType>
            <xsd:choice minOccurs="1" maxOccurs="1">
                <xsd:element ref="CHEM" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="COCIEM" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="CSMX" minOccurs="1" maxOccurs="1"/>
                <xsd:element ref="CLI" minOccurs="1" maxOccurs="1"/>
            </xsd:choice>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="CHEM">
        <xsd:complexType>
            <xsd:attribute name="zonaGeografica" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:token">
                        <xsd:enumeration value="Altiplano Central"/>
                        <xsd:enumeration value="Noroeste"/>
                        <xsd:enumeration value="Noreste"/>
                        <xsd:enumeration value="Golfo"/>
                        <xsd:enumeration value="Sureste"/>
                        <xsd:enumeration value="Oaxaca"/>
                        <xsd:enumeration value="España"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="areaTematica" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:token">
                        <xsd:enumeration value="literatura"/>
                        <xsd:enumeration value="derecho"/>
```

Figura 3.3. Esquema XML para los documentos del Corpus Histórico

En este esquema se puede ver cómo el documento está definido mediante dos elementos, el encabezado y el cuerpo. De hecho, más debajo de las definiciones de los elementos “corpus” y “CHEM” se encuentran las definiciones de “encabezado” y “cuerpo” y todos los elementos que contienen.

Para visualizar o presentar la información de los documentos XML de tal manera que puedan ser leídos como un texto de usuario final, se usan las hojas de estilo ***XSLT***. Una hoja de estilo permite que el autor o el receptor del documento XML puedan definir cómo se tiene que mostrar el contenido del documento. Se puede decir que la hoja de estilo contiene una serie de instrucciones para reescribir el documento y presentarlo de una forma amigable a la vista de la persona que quiera leerlo. Con la hoja de estilo diseñada específicamente para este corpus el Acta de Independencia se visualiza como en la figura 3.4.



Acta de independencia del Imperio Mexicano, pronunciada por su Junta Soberana congregada en la Capital de él en 28 de Setiembre de 1821./

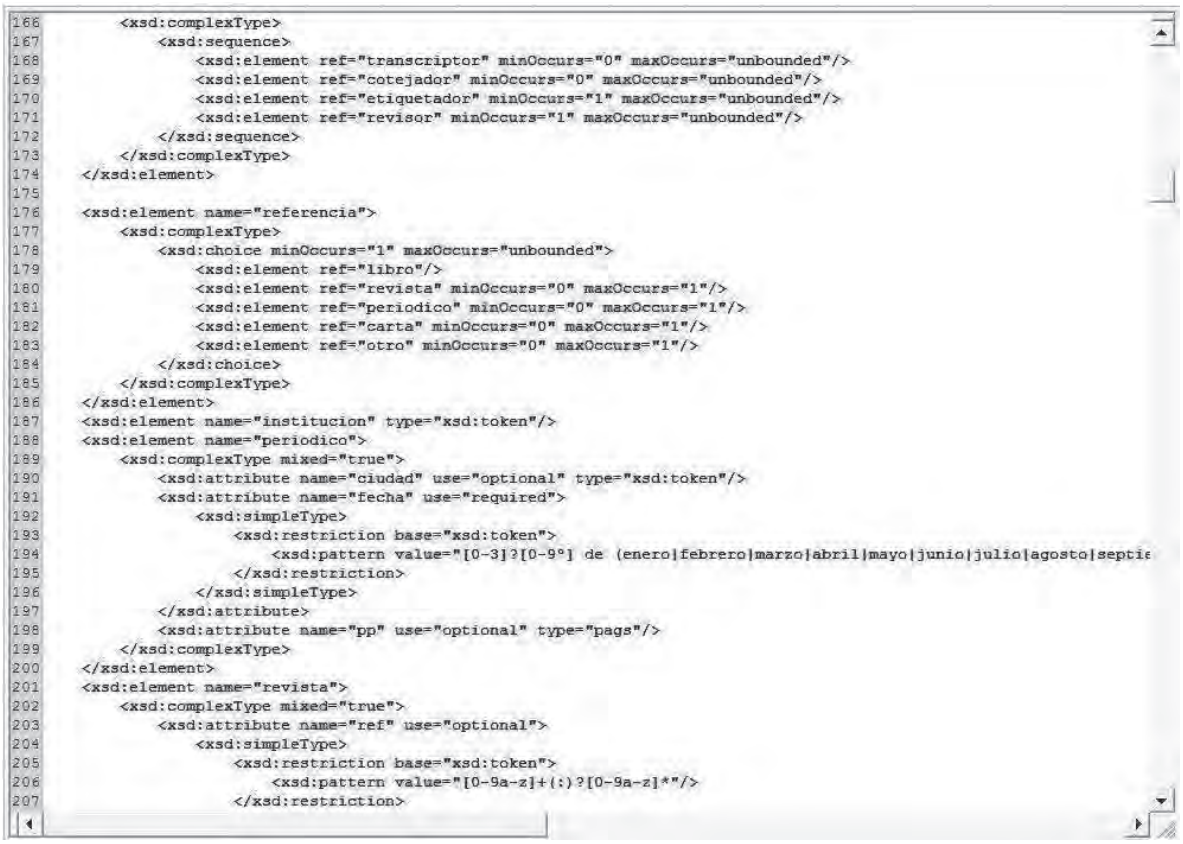
La Nacion Mexicana que, por trescientos años, ni ha tenido voluntad propia, ni libre el uso de la voz, sale/ hoy de la opresion en que ha vivido./

Los heroicos esfuerzos de sus hijos han sido coronados y esta consumada la empresa eternamente memorable/ que un genio, superior á toda admiración y elogio, amor y gloria de su Patria, principio en Iguala, prosiguio y/ llebo al cabo, arrollando obstáculos casi insuperables./

Restituida, pues, cada parte del Septentrion al exercicio de cuantos derechos le concedió el Autor de la na-/ turaleza, y reconocen por inenagenables y sagrados las naciones cultas de la tierra, en libertad de constituirse/ del modo que mas convenga a su felicidad, y con representantes que pueden manifestar su Voluntad y sus desig-/ nios, comienza á hacer uso de tan preciosos dones, y declara solemnemente, por medio de la Junta Suprema del/ Imperio, que es una Nación Soberana é independiente de la antigua España, con quien, en lo sucesivo, no mantendrá/ otra unión que la de una amistad estrecha, en los términos que prescribieren los tratados; que entablará relacio-/ nes amistosas con las demas potencias, executando, respecto a ellas, cuantos actos pueden y estan en posesion de/ executar las otras naciones Soberanas; que va á constituirse, con arreglo á las bases que en el Plan de Iguala y/ Tratado de Córdoba estableció, sabiamente, el primer Gefe del Exercito Imperial de las Tres Garantías, y en fin que/

Figura 3.4. Visualización de documento XML del CHEM (Acta de Independencia)

En este contexto, la tarea del constructor automático de la base de datos de esta tesis es analizar la parte del esquema XML que valida la información bibliográfica de cada documento del corpus. La idea es generar las instrucciones para crear la tabla SQL a partir de este análisis. En el siguiente capítulo, veremos las particularidades de SQL para llevar esto a cabo. Finalmente, esta tabla se puebla con la información específica de cada documento del corpus (lo que se verá en la segunda parte del capítulo 6). En la figura 3.5 se muestra la porción del esquema XML que define el elemento “referencia”, que contiene la estructura de las referencias bibliográficas del corpus.



```
166         <xsd:complexType>
167             <xsd:sequence>
168                 <xsd:element ref="transcriptor" minOccurs="0" maxOccurs="unbounded"/>
169                 <xsd:element ref="cotejador" minOccurs="0" maxOccurs="unbounded"/>
170                 <xsd:element ref="etiquetador" minOccurs="1" maxOccurs="unbounded"/>
171                 <xsd:element ref="revisor" minOccurs="1" maxOccurs="unbounded"/>
172             </xsd:sequence>
173         </xsd:complexType>
174     </xsd:element>
175
176     <xsd:element name="referencia">
177         <xsd:complexType>
178             <xsd:choice minOccurs="1" maxOccurs="unbounded">
179                 <xsd:element ref="libro"/>
180                 <xsd:element ref="revista" minOccurs="0" maxOccurs="1"/>
181                 <xsd:element ref="periodico" minOccurs="0" maxOccurs="1"/>
182                 <xsd:element ref="carta" minOccurs="0" maxOccurs="1"/>
183                 <xsd:element ref="otro" minOccurs="0" maxOccurs="1"/>
184             </xsd:choice>
185         </xsd:complexType>
186     </xsd:element>
187     <xsd:element name="institucion" type="xsd:token"/>
188     <xsd:element name="periodico">
189         <xsd:complexType mixed="true">
190             <xsd:attribute name="ciudad" use="optional" type="xsd:token"/>
191             <xsd:attribute name="fecha" use="required">
192                 <xsd:simpleType>
193                     <xsd:restriction base="xsd:token">
194                         <xsd:pattern value="[0-3]?[0-9°] de (enero|febrero|marzo|abril|mayo|junio|julio|agosto|septi
195                     </xsd:restriction>
196                 </xsd:simpleType>
197             </xsd:attribute>
198             <xsd:attribute name="pp" use="optional" type="pags"/>
199         </xsd:complexType>
200     </xsd:element>
201     <xsd:element name="revista">
202         <xsd:complexType mixed="true">
203             <xsd:attribute name="ref" use="optional">
204                 <xsd:simpleType>
205                     <xsd:restriction base="xsd:token">
206                         <xsd:pattern value="[0-9a-z]+(:)?[0-9a-z]*"/>
207                     </xsd:restriction>
```

Figura 3.5. Porción del esquema XML para los documentos del corpus que define el elemento “referencia”, objetivo principal del constructor.

4. SQL

Este capítulo[3] contiene todo lo relativo a las características de las bases de datos, se menciona la clasificación de los diversos modelos que existen para manejar bases de datos. Se explica de una forma detallada el modelo de base de datos relacional, además de explicar algunos términos como **DBMS** y **RDBMS**. Posteriormente se expone lo que es SQL (en inglés, ***Structured Query Language***) o lenguaje de consulta estructurado, que es un tipo de **RDBMS**. Por último, presentaré los comandos básicos de SQL y cómo se utilizaron para la creación y población del sistema desarrollado en esta tesis.

Base de datos relacionales

Antes de abordar directamente lo que son las bases de datos relacionales y específicamente SQL, hay que mencionar que éstas no son el único tipo de base de datos que existe. A lo largo de la historia se han ido creando diversos modelos o métodos para manejar la información contenida en las bases de datos. Estos modelos, en donde se guarda la información, por lo general no son algo tangible. Más bien son algoritmos y conceptos matemáticos que también nos permiten almacenar y recuperar información.

---

[3] La información presentada en este capítulo se obtuvo de revisar las siguientes fuentes: Momjiam Bruce, *PostgreSQL: Introdcution and Concepts*, Addison Wesley 2001; Douglas Korry, *PostgreSQL*,Sams Publishing; Coles Michael, *Pro T-SQL 2008 Programmer's Guide*, 2008; y PostgreSQL 8.3.1 Documentation. Adicionalmente, se consultaron las siguientes páginas:

- http://www.elserver.com/wiki/SQL
- http://www.htmlpoint.com/sql/sql_08.htm
- http://es.wikipedia.org/wiki/Base_de_datos
- http://www.scribd.com/doc/13664065/007-Bases-de-Datos
- http://arquitecturaestefany.blogspot.com/

Los más comunes son:

- Bases de datos jerárquicos
- Bases de datos reticulares
- Bases de datos de red
- Bases de datos multidimensionales
- Bases de datos relacionales
- Base de datos por objetos

Los dos primeros modelos ya no son muy usados. La mayoría de las bases de datos que se usan en la actualidad pertenecen al tipo relacional. La razón de la popularidad de este tipo es ofrecer la posibilidad de construir sistemas simples y efectivos para representar y manipular los datos. Se basa en relaciones. A un conjunto de datos se les llama tupla. Para explicarlo de una manera más sencilla se podría imaginar que cada relación fuera una tabla que está formada por registros que son las filas o renglones de una tabla, que simbolizarían las tuplas; los campos serían las columnas de la tabla. Como ya se mencionó, todos los modelos de bases de datos se ayudan de algoritmos y conceptos matemáticos. En el modelo relacional se tienen bases teóricas sólidas del álgebra y cálculo relacional, que permite que la información pueda ser recuperada o almacenada fácilmente por medio de comandos llamados “consultas” (***queries***), de forma sistemática y ordenada.

Para manejar y organizar una base de datos se utiliza un **DBMS** (***DataBase Management System***, o sistema de gestión de bases de datos). Estos son un tipo de software que tiene la finalidad de administrar el funcionamiento de la base de

datos, permitiendo almacenar y acceder a la información contenida en dicha base de forma rápida y organizada, ya sean requeridas por un usuario o por alguna aplicación o software que necesite utilizar dicha base para realizar alguna tarea especifica. Ya se mencionaron superficialmente algunas características de las bases de datos que existen y en forma general lo que son los **DBMS,** pero debido a que la tesis tiene como finalidad la creación de una base de datos relacional, se abordará más detalladamente lo que son las bases de datos relacionales, y comenzaré describiendo algunas de sus características.

Una base de datos relacional está compuesta por varias tablas. En ella no pueden existir dos tablas con el mismo nombre, y cada tabla es a su vez, como se dijo arriba, un conjunto de registros, filas o tuplas. Cada uno de estos registros consta de varias columnas, campos o atributos.

Algunas de las restricciones que existen en la creación de este tipo de bases de datos, son que en este tipo no existen dos columnas que se nombren de la misma manera en una misma tabla, los datos que se encuentran almacenados en una columna deben ser del mismo tipo, y todas las filas de una misma tabla tienen el mismo número de columnas. Es importante considerar todas estas restricciones para evitar problemas en el momento de realizar el llenado o poblado de la base de datos.

Para obtener la información que está contenida dentro de una base de datos se utilizan los llamados manejadores de bases de datos relacionales, como ya se mencionó, los que a su vez utilizan un lenguaje relacional para realizar

dichas tareas. Actualmente para realizar todas las consultas relacionadas con las bases de datos se cuenta con dos lenguajes formales, el álgebra relacional y el cálculo relacional. Todas las consultas pueden realizarse aplicando uno u otro método. Sin embargo, puede existir una mayor dificultad si se efectúa una u otra opción. Esto se debe a la forma en que se realiza la consulta. La diferencia más importante es que el álgebra relacional tiene que especificar detalladamente los pasos a seguir para obtener una respuesta o consulta, mientras que en el calculo relacional la respuesta se obtiene sin saber cuál fue el procedimiento para obtener dicha respuesta.

Como se puede deducir, los lenguajes que usan el cálculo relacional son de "más alto nivel" o "más declarativos" que los que se basan en el algebra relacional porque en esta última se va describiendo, aunque sea de una manera parcial, el orden en que se deben seguir las instrucciones. En cambio, en el cálculo relacional el resultado que se desea obtener se introduce a un compilador o intérprete que lo analiza y busca el procedimiento más efectivo para obtener el resultado de la consulta.

Como se puede observar los manejadores de bases de datos son fundamentales para administrar toda la información contenida en ellas, auxiliándose de lenguajes para realizar las consultas. El más utilizado es SQL y debido a que este fue el usado para la creación de los comandos que permitieron la construcción y alimentación de la base de datos descrita en esta tesis, conviene abordar de una manera más detallada sus principales características y algunos de

sus comandos básicos para la creación y población de una base de datos relacional.

Manejadores de base de datos relacionales y SQL

Como se explicó anteriormente, para manejar bases de datos relacionales existen programas especializados. Este software es conocido como **SGBD** (sistema de gestión de base de datos relacional) o, en inglés, **RDBMS** (***Relational database management system***).

El lenguaje más común para construir las consultas de bases de datos relacionales es SQL (como se dijo arriba, ***Structured Query Language***). Se trata de un estándar implementado para los principales motores o sistemas de gestión de bases de datos relacionales.

El modelo SQL necesita que las columnas tengan un orden definido (aunque esto no es necesario en un modelo relacional), gracias a esta característica resulta fácil implementarlo a una computadora, ya que la memoria es lineal.

SQL es un lenguaje que se usa para ejecutar acciones que administran la base de datos. Una de sus características es que mediante el manejo del álgebra y/o el cálculo relacional nos permite realizar consultas con el fin de recuperar información de interés de una base de datos (como ya se explicó arriba), así como también hacer cambios sobre la misma de una manera simple y ordenada.

Los sistemas más populares que utilizan SQL son:

- Oracle
- Sybase
- Microsoft SQL Server
- Ingres
- MySQL
- PostgreSQL

Todos los sistemas tienen comandos especiales que nos permiten administrar de una mejor manera la base de datos (agregados al lenguaje). Sin embargo, con los comandos básicos (***Select, Insert, Update, Delete, Create y Drop***) pueden realizarse la mayoría de las tareas necesarias para administrar y consultar una base de datos.

Otra característica importante que tienen los gestores son que no solamente nos permiten alimentar y consultar datos almacenados en la base de datos, sino que además nos proporcionan herramientas, que garantizan evitar la duplicidad de registros. También garantizan la integridad referencial. Esto quiere decir, que al eliminar un registro se eliminan todos los registros relacionados dependientes entre sí.

Hay que mencionar que a pesar de que las bases de datos relacionales solucionan la mayor parte de la administración de las bases de datos, presentan carencias con datos gráficos (imágenes), multimedia, CAD y sistemas de información geográfica y no se manipulan de forma eficiente los bloques de texto como tipo de dato. Para satisfacer los casos antes mencionados, donde se

presentan ciertas carencias, se crearon las bases de datos orientadas a objetos (**BDOO**) que complementan pero no reemplazan a las bases de datos relacionales.

Creación de bases de datos

Una base de datos consiste en una serie de tablas que tienen relaciones entre sí, que se utilizan para almacenar y obtener información de una manera rápida. La parte fundamental de crear una base de datos consiste en la creación de las tablas que la componen. Sin embargo antes de proceder a la creación de las tablas, normalmente hay que crear la base de datos, lo que a menudo significa definir un espacio de nombres separado para cada conjunto de tablas. De esta manera, para un **SGBD** se pueden administrar diferentes bases de datos independientes al mismo tiempo sin que ocurran conflictos con los nombres que se usan en cada una de éstas. Hay que mencionar que aunque SQL es un estándar, cada **SGBD** que utiliza el lenguaje SQL puede tener un procedimiento propietario para crear una base de datos. Esto quiere decir que además de utilizar los comandos SQL comunes del lenguaje, pueden existir comandos más cortos que tengan la misma función cambiando el nombre de las instrucciones previstas en el estándar. A estos comandos adicionales se les llama agregados al lenguaje.

Por ejemplo, la sintaxis estándar empleada en SQL, que usan todas las **SGBDs** más conocidas, es la siguiente:

CREATE DATABASE nombre_base de datos

Pero además de utilizar la sintaxis arriba mencionada para la instrucción de crear una base de datos, se puede utilizar un agregado de lenguaje propio de PostgreSQL, que aunque se escribe diferente realiza la misma operación que la del estándar SQL. El número de caracteres es menor, lo que nos ahorra tiempo al momento de ejecutar los comandos.

CREATEDB nombre_base de datos

Una vez creada la base de datos, se pueden crear las tablas que la componen. La instrucción SQL para este fin es:

CREATE TABLE nombre_tabla (
nombre_columna tipo_columna [cláusula_defecto] [vínculos_de_columna]
[, nombre_columna tipo_columna [cláusula_defecto] [vínculos_de_columna] ...]
[, [vínculo_de tabla] ...])

Los tipos de datos de las columnas (***tipo_columna***) SQL se clasifican en 13 tipos de datos primarios y de varios sinónimos válidos reconocidos por dichos tipos de datos. Los tipos de datos primarios son:

Binary	***bit***	***byte***	***counter***
currency	***datetime***	***single***	***double***
short	***long***	***longtext***	***longbinary***
text			

Población de la base de datos

Cuando se habla de "poblar la base de datos", se está hablando de la operación de introducir datos en ella. En una base de datos relacional esto corresponde a la

creación de las líneas o tuplas que componen las tablas que forman la base de datos. Normalmente, la memorización de una información concreta corresponde a la introducción de una o más líneas en una o más tablas de la base de datos.

El orden de las operaciones de llenado no es de una manera desordenada. De hecho, la inserción de las líneas tiene que hacerse de modo que se respeten las relaciones formadas en las tablas. De lo contrario, no se podría recuperar la información de una manera correcta.

Hasta aquí, he presentado algunos conceptos básicos relacionados con SQL, que es el lenguaje que utilizaré para crear las bases de datos a partir del análisis de esquemas XML y la posterior población de dicha base, planeadas en esta tesis. En los siguientes capítulos presentaré el desarrollo que seguí para elaborar el constructor y alimentador de la base de datos, comenzando en el capítulo 5 con la metodología seguida.

5. Metodología

En este capítulo[4] se describen los pasos que se siguieron para desarrollar el constructor automático de la base de datos, y la posterior población de dicha base. Se describen todas las especificaciones, requisitos y modelado. Obteniéndose de este análisis los diferentes diagramas UML.

Descripción de lo que se requiere

El objetivo de esta tesis era desarrollar un constructor automático de bases de datos relacionales a partir de esquemas XML para ser pobladas con datos contenidos en los documentos XML. Los pasos que se siguieron para elaborar tal proyecto, se basaron en la metodología de la ingeniería de software, tomando en cuenta las necesidades que se requerían y cuales eran los elementos con que se contaba para tomarlos como base para desarrollar dicho constructor. Se abordaron diferentes aspectos que se tenían que considerar. Para facilitar este análisis se enlistaron en los requisitos las características mínimas que debía tener el programa, y se establecieron como prerrequisitos a los documentos o el sistema con los que se debería trabajar. Enseguida se muestran las listas de los requisitos y prerrequisitos, usados para la construcción y población de la base de datos.

[4] La información presentada en este capítulo se obtuvo de revisar las siguientes fuentes: Alarcón, Raúl, ***Diseño Orientado a objetos con UML***, Eidos 2000; Fowler, Martin, ***UML gota a gota***, Addison Wesley 1999; Pressman Roger, ***Ingeniería del Software***, McGraw-Hill 2002 y Zapata J. Carlos Mario, ***Ingeniería del Software: Una disciplina de modelamiento***, Colombia, 2006. Además se consultaron las páginas siguientes:

- http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf
- http://www.ingenierosoftware.com/analisisydiseno/uml.php

Prerrequisitos

Los prerrequisitos se refieren principalmente a la forma en que se encuentran elaborados los documentos XML. Para construir la base de datos, la idea es tomar la información de su estructura codificada en el esquema XML que se utiliza para determinar si los documentos están bien formados y validados.

Requisitos

Los requisitos se refieren a los resultados que se quieren obtener, tanto para el constructor como para el alimentador, y están resumidos en los siguientes puntos:

- Construir automáticamente una base de datos bibliográfica para el Corpus Histórico del Español en México que está constituido por un conjunto de documentos XML con información variada, además de la bibliográfica.

- A partir de un esquema XML y documentos XML, validados por dicho esquema, obtener la información necesaria para crear una base de datos y posteriormente poblarla.

- No es necesario que se cree la base de datos a partir del nodo inicial del esquema, sino que puede empezar a crearse dicha base a partir de un nodo secundario.

- La base de datos creada será SQL (PostgreSQL)

Desarrollo

Una vez que se definieron cómo eran los tipos de documentos a los cuales se les iba extraer la información (los prerrequisitos) y qué es lo que debería de obtenerse como resultado de dicha extracción de información (los requisitos), se procedió a plantear las mejores alternativas para poder llegar a un resultado óptimo.

Para lograr un resultado adecuado, como metodología utilicé un modelado estructurado, ayudándome de los pasos de desarrollo de software y más específicamente basándome en el lenguaje UML; que aunque no es una metodología formal propiamente, sí proporciona diferentes diagramas que facilitan el diseño.

Para tener una idea de la complejidad del programa a desarrollar, lo primero que hice fue realizar los diagramas de Casos de Uso usando la notación de UML. El Diagrama de Casos de Uso sirve para analizar de una manera general las exigencias que se debe cumplir para realizar el constructor de la base de datos relacional, ya que este tipo de diagramas tiene dos componentes básicos que ayudan mucho en el diseño: los actores que son la representación de las funciones que pueden realizar los diferentes usuarios del sistema y los casos de uso que simbolizan una función especifica que realiza el sistema para solucionar una acción específica. Estos ayudan a reconocer cómo son las interacciones entre los diferentes usuarios o actores y el sistema; todo esto a partir de los análisis de requisitos y prerrequisitos (arriba mencionados).

Hay que mencionar que además de los diagramas de Casos de Uso, también se utilizaron otros diagramas de la metodología UML; que se mencionan a continuación:

- Diagrama de secuencia
- Diagrama de colaboración
- Diagrama de clases
- Diagrama de estados

Estos diagramas se encuentran en el apéndice: Diagramas UML.

Una vez que se logró identificar por medio de los diagramas UML las diferentes rutinas que debería de tener el constructor y el alimentador de la base de datos, el siguiente paso fue analizar las características tanto de los documentos XML y esquemas XML, obteniendo las siguientes conclusiones:

Los documentos XML tienen la información clasificada en piezas de datos delimitadas por etiquetas, aunque existen etiquetas “nulas”(<!—comentario -->), que pueden contener comentarios para facilitarnos como usuarios de los documentos y esquemas XML la identificación de ciertas etiquetas, sin embargo dichas etiquetas “nulas” no son consideradas parte de un documento XML y por lo tanto no representan información útil para la creación y alimentación de la base de datos. También por facilidad en los documentos y esquemas XML se utilizan espacios y tabulaciones para darle un orden a un esquema o documento XML, pero tales elementos (espacios y tabulaciones) no son información pertinente a este trabajo, por lo que hay que ignorarla.

Se asume que los documentos XML que se van a utilizar para alimentar la base de datos ya están bien formados y validados. Como se mencionó en el capítulo de ***Metodología XML***, un esquema XML consiste en las reglas que deben seguir los documentos XML, dichas reglas de un esquema (archivo tipo XSD) nos indican el orden de los elementos, los valores de los datos de atributos y elementos, o la multiplicidad de valores de un documento (por ejemplo elementos que deben repetirse o que no pueden repetirse), etc.

Los documentos XML tienen una estructura jerárquica, en la que existe sólo un único elemento raíz en el que todos los demás están contenidos; y todas las etiquetas están correctamente anidadas una dentro de otra.

Después de analizar y observar los características de los documentos y esquemas XML, el siguiente paso fue analizar cuál era la mejor forma de leer los documentos XML, conocer cuáles eran las posibles opciones para recorrerlos y finalmente el resultado que se podría obtener al realizar el recorrido del esquema para crear la base de datos.

El resultado del análisis que se realizó en este capítulo, tanto de los diagramas UML, como de las características más importantes de los documentos XML, dieron como resultado los diferentes pasos que debería seguir para la elaboración del constructor y alimentador automático de la base de datos.

En el siguiente capítulo se muestra a detalle la forma en que se diseñó el constructor automático de la base de datos, tomando diferentes características a considerar.

6. Construcción de una base de datos a partir de esquemas XML

En este capítulo[5] se muestra el desarrollo que se siguió para elaborar el constructor automático de la base de datos. Apareciendo al principio de este capítulo el algoritmo en el cual se muestra la forma en que debe de comportarse el constructor automático de la base de datos de una forma general, y poco a poco, se vuelve más complejo hasta dar como resultado la obtención de los comandos SQL, necesarios para la creación de la base de datos.

Algoritmo para el constructor automático de la base de datos

Después de que ya se tenía un modelado del constructor automático de bases de datos y se identificaron las características principales que deberían de tener, el siguiente paso fue analizar los posibles casos que se podían presentar al momento de analizar las etiquetas, para este fin se formuló el algoritmo mostrado en la figura 6.1. El algoritmo indica de una manera muy superficial, cómo se debe comportar el constructor automático de la base de datos, al momento de analizar el esquema XML.

Dándonos una idea general del comportamiento general del constructor automático de la base de datos, para posteriormente lograr la identificación de los

[5] La información presentada en este capítulo se obtuvo de revisar la siguiente fuente bibliográfica: MacDonald, Matthew, ***Office 2003 XML for Power Users***, Apress, 2004 y la página http://www.elserver.com/wiki/SQL.

diferentes tipos de etiquetas y cómo clasificar cada una de éstas dependiendo de la información que contengan.

Algoritmo para analizar esquemas XML para construir la bases de datos

1. Recibir los datos de entrada necesarios, para que el constructor automático de la base de datos pueda ser ejecutado correctamente.

2. Preparar las etiquetas para que puedan se leídas correctamente. Quitando comentarios <!---- >, tabulaciones, espacios en blanco, etc. Dejando únicamente la información útil.

3. Al momento de ir leyendo el esquema XML, separar el documento por etiquetas <> y posteriormente almacenarlas en una lista o un arreglo.

4. La etiqueta a partir de la cual se debe iniciar la construcción de la base de datos debe de agregarse a una ***lista*** que busca las tablas y columnas de la base de datos.

5. iniciar el contador con i=0

6. Tomar el elemento 'i' de ***lista***, para analizar

7. Buscar la etiqueta del elemento 'i' con el cual se quiere crear una tabla de la base de datos.

8. El elemento ‘i’, es el nombre de tabla que se desea crear.

9. Buscar cuáles de las etiquetas contenidas dentro de este elemento, son consideradas como columnas de la tabla y cuáles se deben agregar a ***lista***, para analizarse posteriormente y obtener una nueva tabla

10. i=i+1

11. repetir los pasos del 6 al 10 hasta que los elementos contenidos en ***lista***, se acaben.

12. Las instrucciones obtenidas producto de analizar el esquema, enviarlas a un archivo de texto.

Figura 6.1. Algoritmo para analizar esquemas XML

Es importante mencionar que el algoritmo mostrado en la figura 6.1 no analiza todas las posibles estructuras en un esquema XML. Se limita, como se especificó arriba, a procesar las pertinentes para la construcción de la base de datos bibliográfica del Corpus Histórico del Español en México.

En términos generales lo que debe hacer el constructor automático de la base de datos, es agrupar la información del esquema XML para que pueda ser procesada, quitando toda información innecesaria, como pueden ser los comentarios o espacios en blanco. Para analizar dicho esquema, las etiquetas se agrupan en diferentes conjuntos, el ***conjunto1*** contiene todas las etiquetas del esquema XML, dicho conjunto contiene todas las etiquetas del esquema XML, listadas de la misma forma en que aparecen originalmente en el esquema.

Se busca en el ***conjunto1***, si contiene la etiqueta de la forma:

```
<xsd:element name=”elemento_a_buscar”>
```

En caso de encontrar la etiqueta con esas características, el nombre =**”elemento_a_buscar”** se utiliza para nombrar la tabla inicial de la base de datos. A partir de esta y las siguientes etiquetas contenidas dentro de ***conjunto1***, se obtiene la información para la estructura de la tabla de la base de datos. Desde el elemento `<xsd:element name=”elemento_a_buscar”>` hasta su etiqueta de cierre `</xsd:element>`.

Después se espera que dentro de la información almacenada en ***conjunto1*** se encuentren etiquetas del tipo: atributos `<xsd:attribute name="NOMBRE” …>`, y elementos `<xsd:element ref="NOMBRE" …>`. Dichas etiquetas se deben de analizar, principalmente si tales etiquetas son simples o complejas. En caso de ser simples quiere decir que su información se concreta a un tipo de dato (ej. cadena de caracteres, número, un carácter, etc.). Las simples se toman como columnas de la tabla, siempre que el número de apariciones de tal definición sea única. Las complejas, aquellas cuya información está en diversos tipos de datos, corresponden a subtablas, que se van a guardar en una lista, para analizarlas en un proceso recursivo hasta ya no tener más posibles subtablas que analizar.

Una vez que se detectó cuales etiquetas se deben buscar, para iniciar la creación de la base de datos relacional, lo que siguió fue elegir qué estructura de

datos se podría utilizar para almacenar tanto el ***conjunto1***, como la ***lista*** que vaya guardando otras posibles subtablas, y otras que pudieran surgir para guardar nuevas listas.

Se requería, una estructura de datos que fuera dinámica, ya que no se sabe cuál es el número de elementos o etiquetas que se tienen que almacenar y posteriormente ser leídas, y ya que el constructor y alimentador automático de la base de datos se realizaron en el lenguaje de programación de Java. La ***clase*** que mejor cumplía con las condiciones para realizarlos, son la ***clase*** tipo ***Vector***, que por sus características de crecer de forma automática a medida que se agreguen elementos, y adicionalmente nos brinda diferentes herramientas para agregar, eliminar o insertar elementos en una posición especifica, se decidió utilizar dicha ***clase***.

Una vez que se tuvo definido la forma en que debe comportarse el algoritmo y las herramientas básicas que se requerían dentro del lenguaje de programación a utilizar, el siguiente paso, fue realizar un algoritmo mas detallado, que incluyera el comportamiento del constructor automático de la base de datos, dependiendo de los diferentes tipos de etiquetas, y agregando dentro del algoritmo las estructuras tipo ***Vector*** necesarias para guardar las diferentes listas de etiquetas del esquema XML. Llegando al siguiente algoritmo mas detallado que se presenta a continuación.

Los elementos básicos que el constructor automático de la base de datos debe tener como datos entrada, para poder leer el esquema XML y poder construir la base de datos relacional con extensión XSD son:

- El nombre del archivo que contiene el esquema XML, con terminación XSD.
- El nombre de la etiqueta, a partir de la cual se empieza a realizar la construcción de la base de datos.

También es importante que se especifique cuál atributo o etiqueta se considera que es la llave primaria de la primera tabla que se genere, para que a partir de esta, se puede generar una base de datos relacional.

Una vez que ya se tienen los datos necesarios para que el constructor pueda empezar a leer el esquema XML, se debe de ir seleccionando cuál información contenida dentro del archivo es útil y cuál no es necesario considerar, como espacios en blanco, tabulaciones, comentarios, etc. Quedando únicamente la etiquetas del esquema XML.

Estas etiquetas se almacenan en el orden del archivo original, con la diferencia de que se pueden identificar las etiquetas individualmente, y no existen elementos innecesarios. Esta información puede ser consultada o leída, para obtener partes del documento según convenga. Dichas etiquetas se almacenan en ***vector1***.

Adicionalmente se crean otros dos objetos tipo ***vector***, llamado ***vector4*** y ***vector5***. ***vector4*** contiene los elementos que pueden ser considerados como tabla de la base de datos a construir. vector5 va almacenando las instrucciones SQL, producto del recorrido del esquema XML.

El primer elemento que se agrega a ***vector4***, es el nombre de la etiqueta por la cual se quiere empezar a realizar la construcción de la base de datos. Para el ***vector5***, el primer elemento que se crea es que contiene la instrucción de crear la base de datos (aunque puede omitirse si el dato de entrada de la base de datos es nulo).

Una vez que se introdujo la etiqueta a buscar en el ***vector4***, comienza un proceso iterativo, en el que se van extrayendo los elementos contenidos dentro de ***vector4***, en orden ascendente. Empezando por el ***elemento_a_buscar,*** hasta que se agoten los elementos contenidos dentro de dicho ***vector4***.

Como ya se comentó se extrae el primer elemento de ***vector4*** y se busca dentro de las etiquetas contenidas dentro de ***vector1***, la etiqueta que contenga la forma siguiente:

:<xsd:element name="***elemento_ a_buscar***">

Una vez encontrada, se busca si la etiqueta contiene en su parte final autocierre, en caso de ser afirmativo, únicamente se almacena dicha etiqueta en el vector2. En caso de que la etiqueta no presente el autocierre, se obtienen las etiquetas siguientes hasta encontrar la etiqueta de cierre de dicha etiqueta. Almacenándose esas etiquetas en ***vector2***.

A partir de aquí se trabaja con los elementos contenidos dentro de ***vector2***, que contiene la información para crear el nombre de una tabla, las columnas de la tabla de la base de datos, y las posibles nuevas tablas que se pueden generar.

Se pueden obtener 3 casos en el análisis de la información contenida en vector2, los cuales pueden ser que sólo se obtenga una etiqueta, que se obtengan mas de una, y que no se obtengan etiquetas. En caso de no obtener etiquetas, quiere decir que ***elemento_a_buscar*** no se encuentra dentro del esquema XML, y por lo tanto no se pueden crear las tablas de la base de datos.

En caso de que el resultado de la búsqueda dé sólo una etiqueta, se debe buscar si dicha etiqueta es simple o es compleja. Si es simple significa que la información que pretende delimitar, no contiene dentro de esta mas etiquetas. Si es compleja quiere decir que dentro de esta etiqueta contiene otras etiquetas.

Los casos que se pueden presentar, es que su tipo sea de tipo –entero, texto, char, etc-, o que sea compuesto o sea que su tipo se derive en otra serie de

etiquetas. Para ellas se debe de analizar su tipo, si tiene atributos o esta referenciada hacia otras.

Verificando básicamente 3 aspectos, que tengan terminación de texto simple o enteros**,** el número de ocurrencias de las etiquetas, y si se encuentran dentro de el elemento.

En caso de que alguno de los elementos tenga una ocurrencia mayor a 1, agregar el nombre de la etiqueta a vector4.

Para varias etiquetas se debe de revisar si el elemento es ***choice*** o no es ***choice***, ya que si es ***choice***, se debe verificar la máxima ocurrencia de la etiqueta.

Los atributos contenidos dentro de ***vector2,*** generalmente van a ser, columnas de la base de datos, ya que son elementos simples.

Las etiquetas que tienen alguna referencia hacia algo, se deben de buscar si su tipo es de texto simple o números enteros o es una composición de otras etiquetas, en caso de ser simple, pueden ser columnas de la tabla, siempre y cuando su ocurrencia en la tabla sea igual o menor a 1.

En caso de ser ***choice***, se debe de verificar, cuál es el máximo de repeticiones que se puede escoger un elemento. En caso de ser igual a 1, se debe analizar si el número de ocurrencias puede afectar el almacenamiento de los

datos, y si no afecta considerarlo columna de la tabla, en caso contrario agregarlo a ***vector4***, para analizarlo como una nueva tabla.

Al finalizar el análisis de todas las posibles tablas que pudieran generarse y que se encuentran almacenadas en ***vector4***, el siguiente paso es ejecutar los comandos SQL guardados en ***vector5,*** resultado de ir recorriendo las diferentes etiquetas del esquema XML***.***

Comandos SQL para crear la base de datos a partir del algoritmo

Cómo se explicó en la parte de anterior, los comandos SQL necesarios para crear la base de datos se fueron almacenando en ***vector5,*** inferidos al recorrer la estructura del esquema XML. Una vez que se ha recorrido la estructura del esquema en su totalidad, se ejecutan dichas instrucciones. Esencialmente, el comando que se utiliza para la creación de la base de datos es:

```
CREATE DATABASE <NOMBRE_BD>;
```

Que crea la base de datos, donde <NOMBRE_BD>, es el nombre de dicha base. Esta instrucción se inserta en la lista desde el inicio de la ejecución del constructor, ya que se da por hecho que el elemento a buscar se encuentra en el esquema XML y en caso de no encontrarse no ejecutan las instrucciones SQL.

Posteriormente, dependiendo del recorrido del esquema XML, se insertan en la lista comandos de creación de las tablas pertinentes, con los atributos o

elementos que se obtienen del análisis. El tipo de columna puede variar dependiendo de la manera en que esté declarada en el esquema XML (ya sea texto, número entero, etc.).

Así por ejemplo, de la siguiente parte del esquema XML, se obtienen las instrucciones SQL mostradas abajo:

```
<xsd:element name="encabezado">
	<xsd:complexType>
		<xsd:sequence>
			<xsd:element ref="titulo" minOccurs="1" maxOccurs="1"/>
			<xsd:element ref="descripcion" minOccurs="0" maxOccurs="1"/>
			<xsd:element ref="parametros" minOccurs="1" maxOccurs="1"/>
			<xsd:element ref="autor" minOccurs="0" maxOccurs="unbounded"/>
			<xsd:element ref="institucion" minOccurs="0" maxOccurs="1"/>
			<xsd:element ref="referencia" minOccurs="1" maxOccurs="unbounded"/>
			<xsd:element ref="ciudadPublicacion" minOccurs="1" maxOccurs="1"/>
			<xsd:element ref="fechaOriginal" minOccurs="0" maxOccurs="1"/>
			<xsd:element ref="fechaPublicacion" minOccurs="1" maxOccurs="1"/>
			<xsd:element ref="enlace" minOccurs="0" maxOccurs="1"/>
			<xsd:element ref="imagen" minOccurs="0" maxOccurs="1"/>
			<xsd:element ref="responsables" minOccurs="1" maxOccurs="1"/>
		</xsd:sequence>
		<xsd:attribute name="id" use="optional" type="enteroPositivo"/>
		<xsd:attribute name="archivo" use="optional">
			<xsd:simpleType>
				<xsd:restriction base="xsd:token">
					<xsd:pattern value="[0-9a-záäéëíïñóöúü_-]+.xml"/>
				</xsd:restriction>
			</xsd:simpleType>
		</xsd:attribute>
	</xsd:complexType>
</xsd:element>
```

Figura 6.2. Esquema XML

```
CREATE TABLE encabezado (
      id int,
      archivo text,
      titulo text,
      descripcion text,
      institucion text,
      ciudadPublicacion text,
      fechaOriginal int,
      fechaPublicacion int,
      enlace text
);
```

Figura 6.3. Instrucción SQL

Las etiquetas que no aparecen como instrucciones SQL, como ***autor, responsables,*** entre otras, debido a la características en que están definidas en el esquema XML, no se consideran columnas de la tabla que se está creando y por lo tanto deben agregarse a ***vector4***, para posteriormente ser analizadas y así crear las otras tablas pertinentes, hasta que ya no existan otros elementos que deban ser analizados, que se encuentren contenidos en ***vector4,*** y como ya se dijo anteriormente producto de ir recorriendo el esquema XML.

Así finaliza la explicación del funcionamiento del constructor automático de la base de datos. En el siguiente capítulo se describe el funcionamiento del alimentador de la base de datos.

7. Poblar base de datos a partir de documentos XML validados con esquemas XML

En este capítulo se aborda el tema de la población de la base de datos que se creó con el constructor automático de bases de datos. Presentado el algoritmo que identifica los diferentes tipos de etiquetas que pueden aparecer en un documento XML, cómo se crean las instrucciones SQL y finalmente cómo se puebla la base de datos creada con anterioridad.

Población de la base de datos

Una vez que se tenía definida la forma en que funcionaría el constructor automático de la base de datos, el siguiente paso fue crear el alimentador de dicha base, utilizando para esto documentos XML válidos.

Se buscaron diferentes alternativas, buscando la mejor manera en la que ambas partes –constructor y alimentador de la base de datos- pudieran interactuar. Obteniéndose como resultado el algoritmo de la Figura 7.1, que muestra el funcionamiento en forma general del alimentador de la base de datos, con documentos XML.

Algoritmo para analizar los documentos XML

1. Recibir datos de entrada (path de archivos, nodo principal, nombre BD)
2. Acondicionar información, quitando espacios en blanco, comentarios, etc.,

almacenando la información en ***Lista1***

3. Busca elemento de referencia (el mismo que sirvió para crear la BD), en caso de encontrarlos ir a 4; si no, presentar mensaje “no se encontró elemento”
4. En una sola cadena de caracteres obtener la información desde el elemento buscado hasta su etiqueta de finalización

   Ejemplo_Cadena: <elemento><uno>dos</uno></elemento>

5. Una vez obtenida la cadena, revisar etiqueta por etiqueta contenida en la cadena; pueden presentarse las siguientes opciones o tipos de etiquetas.

   a) Si la etiqueta contiene atributos dentro (tipo1)

      <etiqueta atributos=”algo ”… ><otra_etiqueta> …

   b) La etiqueta contiene atributos y tiene autocierre (tipo2)

      <etiqueta atributos=”algo”…/><otra_etiqueta> …

   c) No contiene atributos y después de la etiqueta no hay otra etiqueta (contiene texto) (tipo3)

      <etiqueta atributos=””…>información …

   d) Después de la etiqueta sigue otra etiqueta (tipo4)

      <etiqueta><otra_etiqueta> …

   e) La etiqueta contiene atributos mas información (tipo5)

      <etiqueta atributos=””..>información <…

6. De acuerdo al tipo de etiqueta, la información se extrae de diferente manera, y por lo tanto se crean diferentes comandos SQL, que alimentan la base de datos, almacenándose en una lista llamada ***Lista2***

7. Por ultimo, se revisa si existen varios `INSERT`, para una sola tabla; si sí, ir a 8 , y si no, dejar intacta ***Lista2***
8. Modificar ***Lista2,*** agrupando `INSERT`s, que contengan el mismo nombre de tabla
9. Posteriormente se compara que los resultados obtenidos, producto de recorrer el documento XML, sean congruentes con la base de datos. Haciendo los ajustes mínimos, para que se pueda introducir la información correctamente, ayudándose de las instrucciones que generaron la base de datos contenida, el archivo que creó la base de datos.
10. Crear los comandos SQL, para alimentar la base de datos

Figura 7.1. Algoritmo para analizar los documentos XML

Lo primero que debe hacer al alimentador es recibir los datos de entrada: ***path_archivo*** y ***elemento_raiz o nodo***. El path o ruta indica donde se encuentran el o los archivos XML que van a alimentar la base datos. Y el elemento raíz o nodo, es el mismo que se usó para la creación de la base de datos.

El poblador o alimentador puede tener dos alternativas dependiendo de la información que se ingrese en ***path_archivo***, ya que puede leer varios archivos en un proceso automático o solamente un archivo XML. Para que el alimentador realice la primera alternativa se debe teclear ***"*.xml",*** lo que da como resultado la búsqueda de todos los archivos dentro del directorio del path con extensión XML.

Todos los nombres de los archivos XML se almacenan en una ***lista2***. Si se quiere que el alimentador realice la segunda opción, entonces se ingresa el nombre del ***"documento.xml"*** en donde ***documento*** es el nombre de un archivo con extensión XML, lo que almacena sólo un elemento a la ***lista2***. En caso de encontrarse el o los archivos XML, el siguiente paso es leer la información contenida dentro de dicho archivo. A continuación se explica con detalle.

La lectura del archivo XML que posteriormente poblará la base de datos, comienza preparando la información para que pueda ser manejada correctamente, quitando tabulaciones, espacios en blanco innecesarios, comentarios, y todo aquello que puede provocar un error en la correcta lectura del documento XML. El poblador barre el documento buscando el elemento raíz especificado en la línea de comando. El contenido de este elemento (etiqueta: ***<elemento_raíz … >***), que puede ser una estructura compleja se concatena en una sola cadena de caracteres; termina de agregar etiquetas a la cadena al encontrar la etiqueta de cierre del elemento raíz (***… </elemento_raiz>***). Una vez que todo está unido en una sola cadena de caracteres, el alimentador busca el inicio y final de cada etiqueta, contenida dentro de la cadena, y dependiendo de la información que contenga cada etiqueta y los caracteres que sigan después del fin de la etiqueta (puede ser el inicio de otra '<' o caracteres diferentes a este). El alimentador se comporta de acuerdo al algoritmo (Figura 7.1), dando como resultado la creación de los comandos SQL correspondientes para poblar la base de datos, almacenándose en una lista llamada ***ListaSQL***.

Por último, el alimentador hace una revisión dentro de las instrucciones SQL generadas para verificar que no existan comandos *INSERT INTO* con el mismo nombre de la tabla (*INSERT INTO tabla1(…), INSERT INTO tabla1(…)*). Esto es importante porqué nos evita que se añada la información a la base de datos de manera incorrecta y la información que debería estar en una sola fila, se agregue en diferentes filas de la tabla, provocando con esto errores en el llenado de las tablas de la base de datos.

Para solucionar este problema, en caso de encontrar varios comandos *INSERT INTO tabla (…)* con el mismo nombre de tabla, el alimentador las une. A continuación se muestra un ejemplo donde al principio aparecen varios inserciones a una tabla de la base de datos y después de hacer la unión de los comandos SQL, queda en una sola instrucción.

Ejemplo

sin unir comandos SQL

```
INSERT INTO encabezado(id,permisos)
VALUES ('0','todos');
INSERT INTO encabezado(titulo)
VALUES ('Acta de independencia del Imperio Mexicano');
INSERT INTO encabezado(institucion)
VALUES ('Junta Soberana del Imperio Mexicano');
INSERT INTO encabezado(ciudadPublicacion)
VALUES ('ciudad de México');
INSERT INTO encabezado(fechaPublicacion)
VALUES ('1821');
```

después de unir comandos SQL

```
INSERT INTO encabezado(id, permisos, titulo, institucion,
ciudadPublicacion, fechaPublicacion)
VALUES('0','todos','Acta de independencia del Imperio Mexicano','Junta
Soberana del Imperio Mexicano','ciudad de México','1821');
```

Por último, el alimentador de la base de datos ejecuta las instrucciones para poblar la base de datos. En este contexto debo señalar lo siguiente: la única diferencia entre leer uno o varios archivos es que en la primera sólo hay una iteración (porqué ***lista2*** contiene sólo un elemento***)*** y para más de un archivo las iteraciones son igual al número de archivos almacenados en la ***lista2*** que contiene los nombres de todos los archivos XML que alimentarán la base de datos. La iteración comienza en la sección donde se acondiciona (quitar tabulaciones, etc., y hacer la cadena de caracteres) la información de los documentos XML y termina en la revisión de duplicidad de instrucciones.

Hay que mencionar que en el caso de que se quiera leer varios archivos ***(*.xml)***, el ***elemento_raíz,*** o ***nodo***, es el mismo para todos los archivos que van a poblar la base de datos y si no se encontraran los archivos XML aparecería un mensaje en donde se indica que no se encontró el archivo XML. Y por último, en caso de omitirse la extensión '.XML', al momento de escribir el nombre del o los archivos, el alimentador de la base de datos muestra un mensaje que indica "sólo se usan archivos XML para alimentar la base de datos".

Así concluye la explicación del funcionamiento de alimentador de la base de datos. El siguiente capítulo trata de las conclusiones finales, donde se realiza una síntesis de los capítulos anteriores, se abunda sobre las ventajas, desventajas de las decisiones tomadas y se mencionan otras opciones que pudieran utilizarse para consultar documentos XML.

8. Conclusiones finales

En el presente trabajo, lo que hice para elaborar el constructor fue primero investigar lo que eran los esquemas XML; luego, visualizar la posible solución de un caso en particular y; finalmente, decidir cuál era el comportamiento que tenía que seguir el constructor cuando encontrara las diferentes etiquetas dentro del esquema, lo que forma una idea aproximada de cómo debería ser la base de datos que se quería construir.

Después de saber cuáles eran las posibles opciones que se podían seguir, se buscó saber cuáles eran las diferentes formas en que se podría presentar la información dentro del esquema XML; como por ejemplo, una etiqueta podría empezar en un renglón y terminar en el siguiente. El constructor tendría que reconocer cuáles eran esos casos y juntar la etiqueta en un solo renglón. También era importante eliminar tabulaciones, entre otras marcas de espacio en blanco, y substituirlas por espacios en blanco.

Sabiendo todo lo anterior, hice un análisis y obtuve una idea general de cómo debería estar estructurado el programa, y decidí qué clases se necesitaban crear para poder realizar el constructor de la base de datos, ayudado de los diagramas UML y el modelado estructurado.

Lo que siguió fue, tomando como referencia el recorrido que se hizo de un esquema XML (que se mencionó al principio), hacer el análisis más general y saber cuáles eran las posibles decisiones que podría tomar el constructor, tanto en

la lectura de los datos del mismo archivo de texto, como en las decisiones que debe tomar para crear las tablas. Fue también importante encontrar una forma para que el proceso sea recursivo y avanzar en la estructura del esquema para crear las tablas y subtablas para, finalmente, crear los comandos SQL, ejecutarlos y crear la base de datos.

Las desventajas de todo esto son que no se abarcan todos los posibles casos que puede haber al momento de crear un esquema XML y un documento XML, debido a que el desarrollo es específico para el Corpus Histórico de Español en México. La otra desventaja importante es que se generan algunas tablas vacías. De hecho, con estas tablas vacías la normalización de la base de datos se vuelve muy difícil.

Lo importante es que con este desarrollo se pueden crear bases de datos, generando simples reglas a partir de un esquema XML. Lo cual ayuda especialmente en caso de no tener muchos conocimientos acerca de este tema, y se cuente ya con una gran cantidad de documentos XML, y también que las instrucciones generadas con el constructor se puedan ejecutar en cualquier gestor de base de datos PostgreSQL, o SQL.

El desarrollo del trabajo descrito en esta tesis soluciona un problema específico de los corpus electrónicos del Instituto de Ingeniería, específicamente del Corpus Histórico del Español en México. Como trabajo futuro y ya que estos corpus están constituidos por documentos XML, valdrá la pena explorar la

consulta de los datos de estos documentos, especialmente los bibliográficos, mediante las estrategias desarrolladas específicamente para este tipo de documentos; esto es, la aplicación de las tecnologías relacionadas con xQuery que no requieren la construcción adicional de una base de datos. Cabe mencionar que incluso PostgreSQL ya cuenta con comandos específicos para consultas en documentos XML.

Apéndice: diagramas UML

Este apéndice contiene los diagramas UML, tanto del Constructor como del Alimentador de la base de datos.

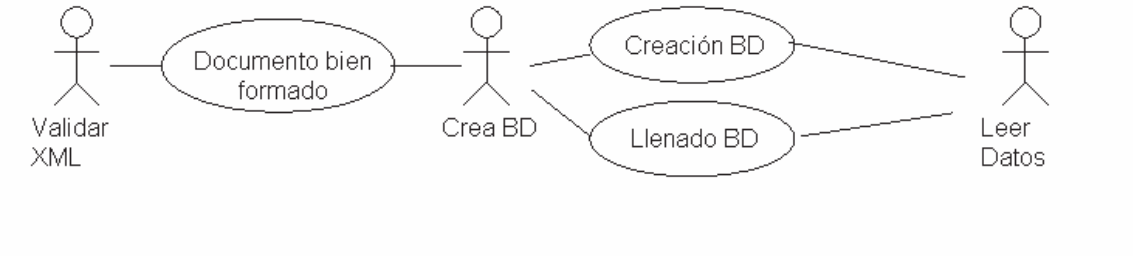


Diagrama 1. Diagrama de Casos de Uso

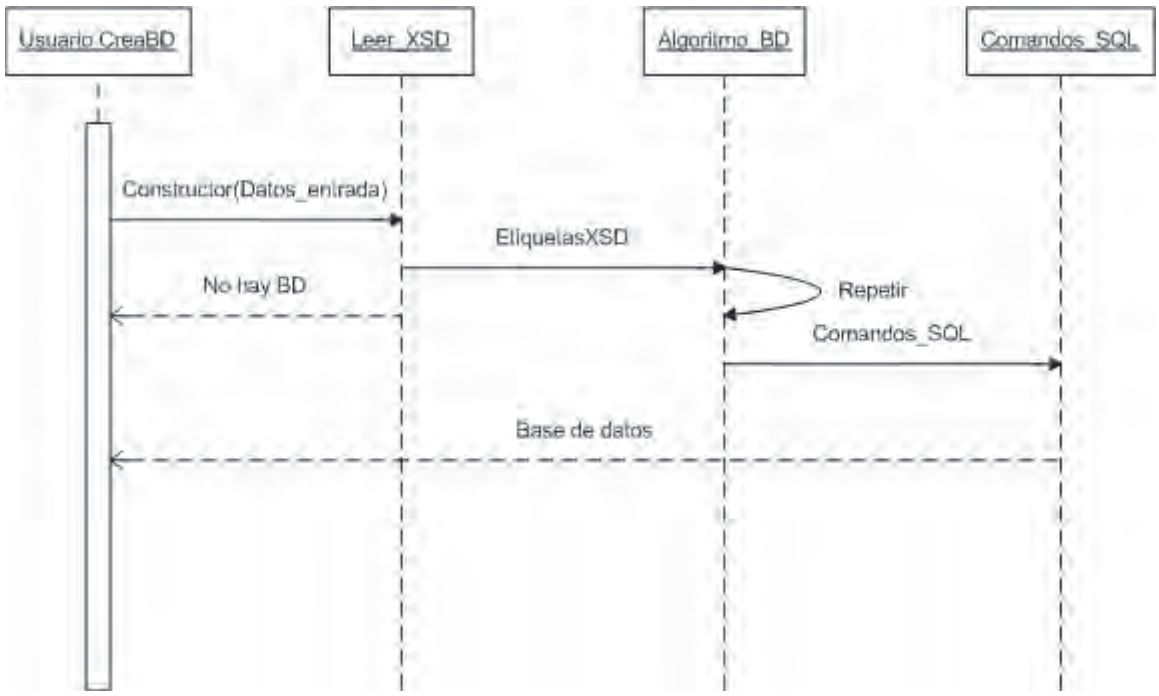


Diagrama 2. Diagrama de Secuencia para el Constructor de la base de datos

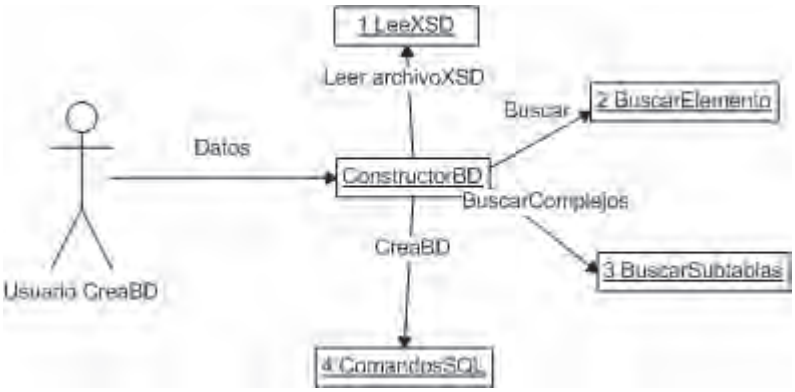


Diagrama 3. Diagrama de Colaboración del Constructor de la base de datos

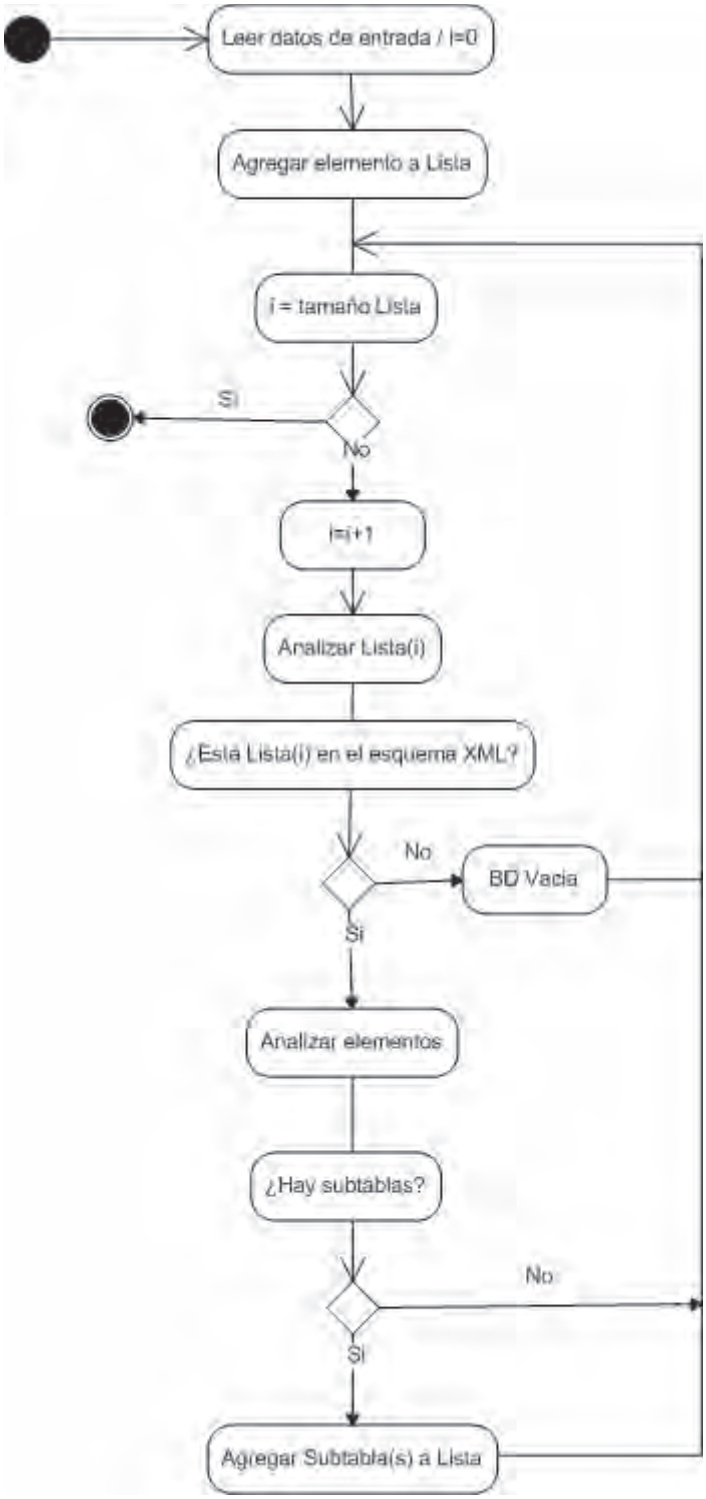


Diagrama 4. Diagrama de Actividad para el Constructor de la base de datos

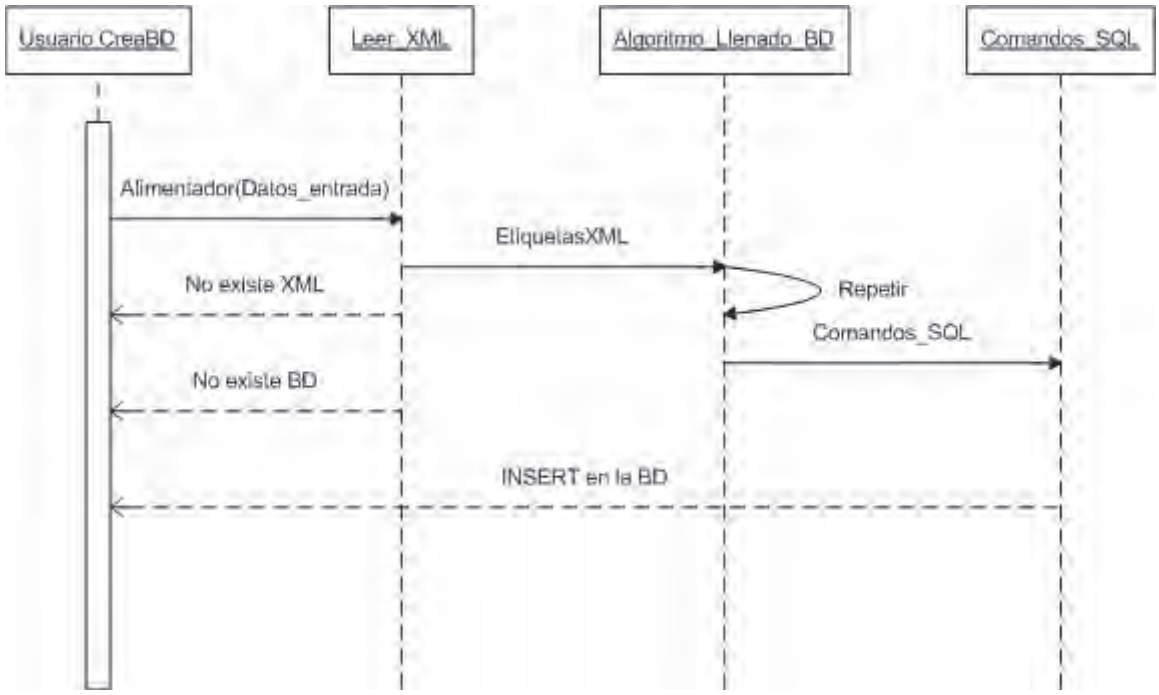


Diagrama 5. Diagrama de Secuencia para el Alimentador de la base de datos

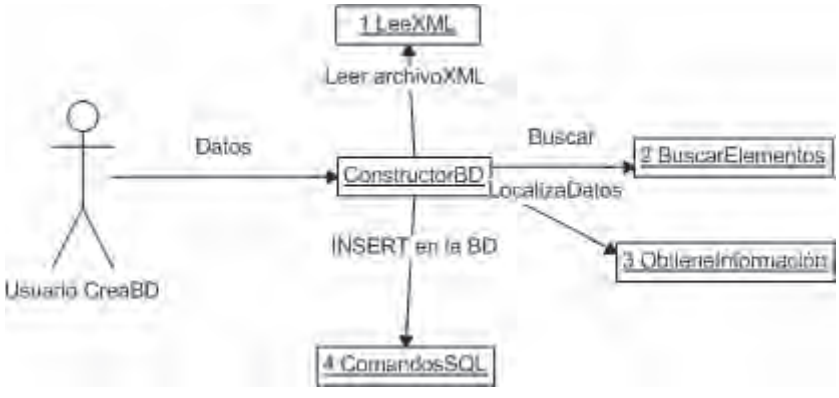


Diagrama 6. Diagrama de Colaboración para el Alimentador de la base de datos

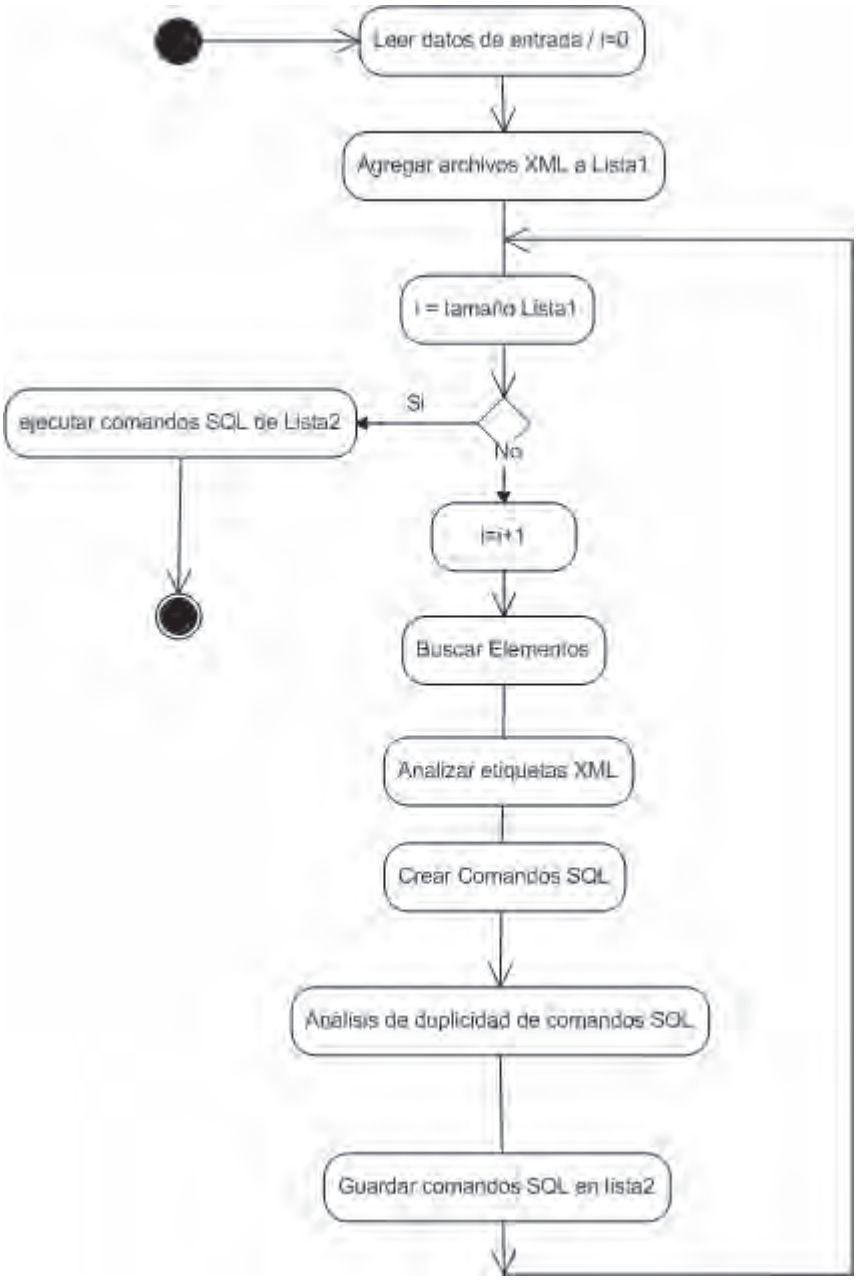


Diagrama 7. Diagrama de Actividad para el Alimentador de la base de datos

Constructor automático de la base de datos	
Nombre:	Constructor automático BD
Autor:	Axel Gerardo Garduño Sandoval
Fecha:	03/04/2009
Descripción: Se pretende Crear una base de datos a partir de esquemas XML	
Actores: Validador XML, Contructor BD, Analizador BD.	
Precondiciones: Los Documentos XML , deben de estar validados y bien formados	
Flujo Normal: 1. El constructor BD, recibe el documento validado 2. Se ejecuta el programa, poniendo los datos (ubicación del archivo con extensión XSD, NOMBRE_BD, NODO_BUSCAR) 3. El programa realiza un algoritmo para identificar los diferentes tipos de etiquetas que existen en el Esquema XMl 4. Crea la base de datos	
Flujo Alternativo: 4. No se encuentra el archivo o el nodo a analizar, sale del programa	
Poscondiciones: Se crea una base de datos con comandos SQL	

Figura A.1. Ficha del Constructor automático de la base de datos

Alimentador automático de la base de datos	
Nombre:	Alimentador automático BD
Autor:	Axel Gerardo Garduño Sandoval
Fecha:	03/04/2009
Descripción: Se pretende alimentar base de datos creada por "El constructor automatico de BD"con documentos XML	
Actores: Validador XML, Contructor BD, Analizador BD.	
Precondiciones: Los Documentos XML , deben de estar validados y bien formados	
Flujo Normal: 1. El alimentador BD, recibe el documento XML validado 2. Se ejecuta el programa, poniendo los datos (ubicación del archivo con extensión XML, NOMBRE_BD, NODO_BUSCAR) 3. El programa realiza un algoritmo para identificar los diferentes tipos de etiquetas que existen en el documento XML y el contenido de cada una de las etiquetas 4. Alimenta la base de datos	
Flujo Alternativo: 4. No se encuentra el archivo o el nodo a analizar, realiza la alimentación en el siguiente archivo o sale	
Poscondiciones: Se crean comandos SQL para alimentarla base de datos	

Figura A.2. Ficha del Alimentador automático de la base de datos

Definiciones pertinentes al algoritmo “*Constructor de la base de datos*”

Este cuadro de definiciones considero oportuno ponerla para lograr una mejor explicación y sirva de referencia, para la explicación del algoritmo de la creación de la base de datos.

Al momento de ejecutar el programa (constructor automático de la base de datos), debe haber 3 cadenas de caracteres de entrada que son:

- “e***lemento_a_buscar***”, que indica a partir de cual elemento se quiere formar la base de datos,
- “***Ruta_ esquema_XML***”, este dato de entrada nos dice la ruta o path, en donde se encuentra almacenado el esquema .xsd.
- ”***Llave_primaria***”, nos informa cual es el atributo que contiene información única e irrepetible que nos ayudará a recuperar la información contenida en la base de datos.

  Descripción de las Listas llamadas Vector:

- ***Vector1***: Contiene todas las etiquetas del esquema XSD
- ***Vector2***: Contiene las etiquetas `<xsd:element name=”elemento_a_buscar”>` hasta su etiqueta de

cierre `</xsd:element>`.

- ***Vector4***: Contiene los ”elemento_a_buscar” o nodos a buscar. Cada nodo se escribe NODO(i) donde ***i*** es un número, el NODO(1), es el ”elemento_a_buscar”, los siguientes NODOS(), son los que se encuentre al recorrer el esquema XSD.
- ***Vector5***: Contiene todas las instrucciones SQL guardadas, producto de ir recorriendo el esquema XSD.

Figura D.1. Definiciones correspondientes al algoritmo del constructor automático de la base de datos

Bibliografía

Artículos y libros

MacDonald, Matthew, ***Office 2003 XML for Power Users***, Apress, 2004.

Alarcón, Raúl, ***Diseño Orientado a objetos con UML***, Eidos 2000.

Fowler, Martin, UML gota a gota, Addison Wesley 1999.

Pressman Roger, ***Ingeniería del Software***, McGraw-Hill 2002

Zapata J. Carlos Mario, ***Ingeniería del Software: Una disciplina de modelamiento***, Colombia, 2006.

Momjiam Bruce, ***PostgreSQL: Introdcution and Concepts***, Addison Wesley 2001

Douglas Korry, ***PostgreSQL***,Sams Publishing.

Coles Michael, ***Pro T-SQL 2008 Programmer's Guide***, 2008.

Burd Barry, ***Beginnning Programing with Java For Dummies***, Wiley Publishing 2005

PostgreSQL 8.3.1 Documentation

Páginas electrónicas

http://es.wikipedia.org/wiki/XML

http://es.wikipedia.org/wiki/Validacion_xml

http://manuales.dgsca.unam.mx/xml/Qu%E9%20es%20DTD.htm

http://www.sidar.org/recur/desdi/traduc/es/xml/xml10p/xml10p.htm

http://www.w3.org/TR/xml11/

http://es.geocities.com/alba1509/Fase3/teoria.html

http://www.scribd.com/doc/6974950/XML

http://www.webtaller.com/construccion/lenguajes/xml/lecciones/xml-10-puntos.php

http://www.elserver.com/wiki/SQL

http://www.htmlpoint.com/sql/sql_08.htm
http://es.wikipedia.org/wiki/Base_de_datos
http://www.scribd.com/doc/13664065/007-Bases-de-Datos
http://arquitecturaestefany.blogspot.com/
http://es.tldp.org/Postgresql-es/web/navegable/tutorial/x574.html
http://es.tldp.org/Tutoriales/doc-modelado-sistemas-UML/doc-modelado-sistemas-uml.pdf
http://www.ingenierosoftware.com/analisisydiseno/uml.php