XMI
Una comparación entre distintas estrategias de almacenamiento relacional, indexación y consulta para documentos XML

Tesis de Licenciatura

Universidad de Buenos Aires
Facultad de Ciencias Exactas y Naturales
Departamento de Computación

Celina Di Blasi
Horacio A. Gonzalez

Resumen

Introducción

XML

En 1986 la ISO (asociación internacional de estándares) aprobó la especificación de un lenguaje descriptivo de anotaciones, denominado SGML. Diez años después, en 1996, el Word Wide Web Consortium (WC3) lanza el SGML para la web, como primer antecesor inmediato del lenguaje XML. En marzo de 1997, las tecnologías asociadas a XML como así también el propio lenguaje es aceptado por Microsoft, el gigante de la informática y rápidamente incorporado a sus productos. En Febrero de 1998 se presenta la especificación final del lenguaje XML, la versión 1.0.

La anterior es una resumida reseña de la aparición del lenguaje XML. En poco más de 3 años, lo que fuera una simple especificación de un organismo de internet se ha convertido en uno de los campos más promisorios en lo que se refiere a la comunicación de datos, la distribución de procesos y el almacenamiento de información.

Qué es XML

El lenguaje XML (eXtensible Markup Language) es un meta-lenguaje que provee un formato para describir datos junto con su estructura. Está diseñado fundamentalmente para el intercambio de información entre sistemas, dispositivos y redes de información, entre las que se destaca la Internet.

Es similar en su estructura al lenguaje HTML (Hiper Text Markup Language) en cuanto al uso de *tags* (etiquetas) y atributos, pero difiere del mismo ya que en XML los tags solo delimitan los datos, no interpretan los mismos, dejando esto para la aplicación que los utiliza.

XML es un lenguaje extensible, ya que permite la definición ilimitada de *tags* y *atributos* dentro de un documento. Los documentos XML deben seguir una estructura jerárquica en lo que respecta a los *tags, un tag* debe estar incluido correctamente en otro. Un documento XML contiene datos que se autodefinen. Se separa el contenido de la presentación de forma total.

El siguiente es un ejemplo de un reporte meteorológico expresado en formato XML:

```
<weather-report>
   <date>March 25, 1998</date>
   <time>08:00</time>
   <area>
      <city>Seattle</city>
      <state>WA</state>
      <region>West Coast</region>
      <country>USA</country>
   </area>

   <measurements>
      <skies>partly cloudy</skies>
      <temperature>46</temperature>
      <wind>
```

```
         <direction>SW</direction>
         <windspeed>6</windspeed>
      </wind>
      <h-index>51</h-index>
      <humidity>87</humidity>
      <visibility>10</visibility>
      <uv-index>1</uv-index>
   </measurements>
</weather-report>
```

El lenguaje XML es libre de licencia, independiente de la plataforma informática y de amplio soporte por parte de la industria y de organizaciones académicas y científicas. Sus lineamientos generales se encuentran regidos por la W3C (World Wide Web Consortium), que es una entidad encargada de crear estándares para la Web. Además de la especificación XML 1.0 que define que son los "tags" y "atributos", existen una serie de tecnologías relacionadas, que complementan el lenguaje otorgándole mayor potencia y flexibilidad. Entre ellas podemos mencionar XLink que agrega hiperlinks en documentos XML, XSL para agregar plantillas de estilo en la visualización de documentos y XSLT para la transformación de documentos.

Necesidades

Hoy en día es necesario poder intercambiar grandes volúmenes de datos entre sistemas incompatibles. Por ejemplo, una empresa que necesite volcar la misma información en diferentes sistemas debería declarar sus estructuras y datos en

diferentes formatos, replicando los datos y poder llevar a que los mismos sean incompatibles en los diferentes sistemas.

XML un lenguaje que permite en forma eficaz transportar datos entre sistemas incompatibles en un formato neutral. XML puede manejar distintos tipos de datos, incluyendo texto, imágenes, sonido y puede extenderse para cualquier otro tipo de datos.

Usos del lenguaje

En muchas áreas, como salud, gobierno y finanzas, matemática, biología y química entre otras, son utilizadas aplicaciones XML para el almacenamiento y procesamiento de datos. XML es un simple método de representación y organización de datos, que lleva a que los problemas de incompatibilidad de datos sea más manejable.

En el campo de Ingeniería Química se generó una nueva aplicación XML llamada Molecular Dynamics Language (MoDL). MoDL provee simples construcciones llamados atom, molecule y bond. Estas construcciones se definen como tags para representar los átomos, moléculas y enlaces respectivamente. El objetivo es poder simular estructuras químicas.

En Matemática se creo la aplicación MathML para describir notación matemática y captura la estructura y contenido. El objetivo de MathML es permitir que

construcciones estructuras matemáticas puedan ser utilizadas, recibidas y procesadas en la Internet.

En el campo de Matemática Discreta se creó la aplicación XGMML (eXtensible Graph Markup and Modeling Language) basada en GML utilizada para la descripción de grafos. XGMML utiliza tags para describir los nodos y los ejes de un grafo. El propósito de XGMML es hacer posible el intercambio de grafos entre diferentes herramientas gráficas. Utilizando XGMML es posible convertir grafos a diferentes formatos.

El lenguaje XML también se está convirtiendo rápidamente en un standard para el almacenamiento e intercambio de información biológica. Iniciativas como el formato GAME (Genome annotation markup elemets) dan muestra de esto. También tiene un gran impulso la creación de interfases XML para las mayores bases de datos de información biológica (secuencias de ADN en general), entre las que mencionamos GenBank y Swiss-Prot.

Iniciativas

En función de potenciar aún más el lenguaje XML, grupos de trabajo pertenecientes a la entidad W3C están trabajando en nuevos protocolos.

Entre ellos podemos mencionar al equipo de trabajo XML Query. El mismo busca proveer mayor flexibilidad a la extracción de datos desde documentos reales y virtuales en la Internet.

El grupo de trabajo XML Linking está diseñando vínculos hipertextos para documentos XML. El grupo tiene como objetivo diseñar hipervínculos avanzados, mantenibles, y de gran escala.

En cuanto a la definición de tipos de documentos marcados, el grupo de trabajo XML Schema busca definir restricciones sobre como deben interactuar las partes de una misma componente. El objetivo de este grupo es definir reglas para las estructuras, contenido y la semántica de documentos XML.

Otra iniciativa importante para mencionar en la cualel lenguaje XML juega un papel fundamental es SOAP (Simple Objetc Access Protocol). Se trata de una especificación para proveer servicios de software a través de la Internet. Estos servicios estarán disponibles para todo aquel que tenga acceso a la web y las credenciales de seguridad adecuadas. Para el intercambio de información entre el cliente y los servidores que albergan a los "Web Services" se utilizan mensajes en formato XML.

Revisar http://www.rpbourret.com/xml/XMLAndDatabases.htm (lo tengo impreso)

XML comenzó siendo un formato utilizado en la web para el intercambio de datos dentro de la web. En los últimos tiempos, XML se hizo muy popular y es utilizado tanto en web como fuera de la ella para el intercambio de datos, y almacenamiento de datos en sistemas orientados a objetos. *Estos documentos XML contienen diferentes tipos de datos, como ser relacionales, long string, textos, etc y en muchos casos es necesario acceder a la información directamente desde el archivo para lo que es necesario almanecarlo y consultarlo en forma rápida y sencilla. Este almacenamiento puede ser muy variado, como ser un file system, bases de datos relacionales, bases de datos orientadas a objetos (Excelon), o sistemas semi-estructurado como Lore(Stanford), Lotus Notes, or Tamino*

Mapear los datos del documento XML en la base de datos se puede realizar utilizando el DTD *(poner el significado de las siglas)* del documento XML, o sin la necesidad del DTD. En el primer caso es necesario que éste exista para poder realizar su almacenamiento, evita controlar que el documento esté bien formado y basta con leer el DTD para reconocer la estructura del documento. En el segundo caso, es necesario leer todos los datos en un proceso de parser para chequear que sea un documento XML bien formado y para inferir la estructura del mismo. Si bien el parsear el documento para inferir su estructura es un proceso más costoso y complejo que el leer el DTD del documento, permite que se puedan almacenar todos los documentos XML, ya que no siempre existe un DTD para leer y poder inferir su estructura.

En este trabajo decidimos no utilizar DTD y si parsear los documentos XML para inferir su estructura y cargar los datos, cabe aclarar que en este trabajo no ponemos énfasis en la carga del documento XML en las bases de datos relacional sino en las consultas a los datos y en la reconstrucción de los documentos almacenados.

Para el parseo utilizamos DOM

El propósito de este trabajo es comparar diferentes modelos de bases de datos relacionales utilizados para almacenar la información de documentos XML, evaluando tiempos de respuestas en diferentes consultas, relación entre tamaño del documento XML y el tamaño de la base de datos al almacenarse el documento, reconstrucción desde la base de datos del archivo almacenado.

Los modelos propuestos para este trabajo son “Mapping Edges” (mapeo por ejes), *“Mappin Paths”* (Toxin), y un modelo propuesto por nosotros llamado XMI.

Mapping Edges un simple esquema para almacenar todos los ejes de un grafo que representa un el documento XML en una tabla simple, que llamaremos tabla *Eje*. Esta tabla guarda los pares de objetos (origen-destino de cada uno de los ejes del grafo), el nivel del eje, un flat para indicar si ese objeto es un nodo interno o un nodo de valor (una hoja).

La estructura de la tabla es la siguiente:

Eje (source, ordinal, name, flag, target)

Ver como es nuestra estructura de la tabla para este modelo porque Florescu presenta más de un modelo y los compara entre ellos, aclarar si hay índices, etc.

El trabajo está organizado en la siguiente manera: en el Capítulo 1 describe los modelos de indexación por path y por ejes, que serán comparados con el modelo propuesto en este trabajo XMI. El Capítulo 2 describe el modelo XMI propuesto en este trabajo, etc….En el Capítulo 3 se presentan las pruebas realizadas sobre los modelos de Path, Ejes y XMI. Finalmente en el Capítulo 4 se muestran y analizan los resultados de diferentes consultas realizadas con los tres modelos, como así también la reconstrucción de documentos.

En el Apéndice A se muestran detalles de la implementación del modelo XMI. El Apéndice x…

Donde especificamos el tema de los índices?

Objetivo

El objetivo de nuestro trabajo será proveer mecanismos eficientes para el acceso rápido a información contenida en documentos XML de gran volumen. Estos mecanismos deberán permitir el acceso a la información tanto para su consulta como para la modificación o agregado de datos.

Algunas de las características más importantes del lenguaje XML, como su estructura jerárquica y el anidamiento de datos hacen particularmente complicada la indexación de datos tal como se la conoce desde tiempo atrás dentro de las bases de datos tradicionales (relacionales). Para solucionar esta cuestión es que nos proponemos especificar, almacenar y administrar documentos XML indexados a través de otra estructura XML, a la que denominaremos iXML. Los documentos iXML serán entonces documentos de indexación aplicables a un determinado documento XML que servirá como fuente de datos. Los documentos iXML seguirán una especificación diseñada para facilitar su almacenamiento y su uso extensivo por todo aquel que posea el documento XML que fue originalmente indexado.

Para un determinado documento XML podrán existir múltiples documentos iXML, cada uno de ellos indexando los datos de manera particular, por ejemplo, para determinados niveles de anidamiento o para todos los datos contenidos en un mismo nivel particular.

A su vez, los documentos iXML compartirán todas las ventajas expuestas por sus pares XML, mencionando por ejemplo su facilidad para ser transmitidos a través de

redes amplias de datos como la Internet y su correcto funcionamiento en múltiples plataformas. Como única limitación encontramos que los documentos XML deberían estar guardados en nuestro motor de almacenamiento para el acceso más eficiente a los datos del mismo.

Capítulo I - Modelos para el almacenamiento e indexación relacional de documentos XML

Uno de los problemas que han surgido hasta el momento es el manejo de los datos etiquetados XML. La búsqueda, análisis, actualización y selección de datos pueden ser procesadas en un ambiente manejable, sistemático y comprensible, las bases de datos. Una de las ventajas de utilizar esta estructura es la familiaridad que tiene el usuario con ellas, son bien conocidas y confiables.

Tanto las bases relacionales como las bases orientadas a objetos se pueden utilizar para almacenar documentos XML. También se crearon nuevas bases de datos XML, que pueden almacenar los datos.

Trasladar datos XML desde y hacia una base de datos requiere tiempos considerables, especialmente para documentos de gran volumen. Este factor puede resultar en algunos casos molestos cuando se relacionan datos de más de un documento XML.

Para acceder de forma eficiente a la información almacenada tanto para actualización como inserción de datos, se utilizan índices. Los índices son una buena herramienta para localizar datos particulares dentro de estructuras con grandes volúmenes de información.

Mapping paths

Ver tesis de Toronto

Mapping edges

Ver “Storing An Quering XML Data” de Florescu
Acá puede ir algo de esto, no se si la comparación

XMI

Fundamentos

Se representa el documento XML con una estructura de árbol.

El árbol tendrá dos tipos de niveles

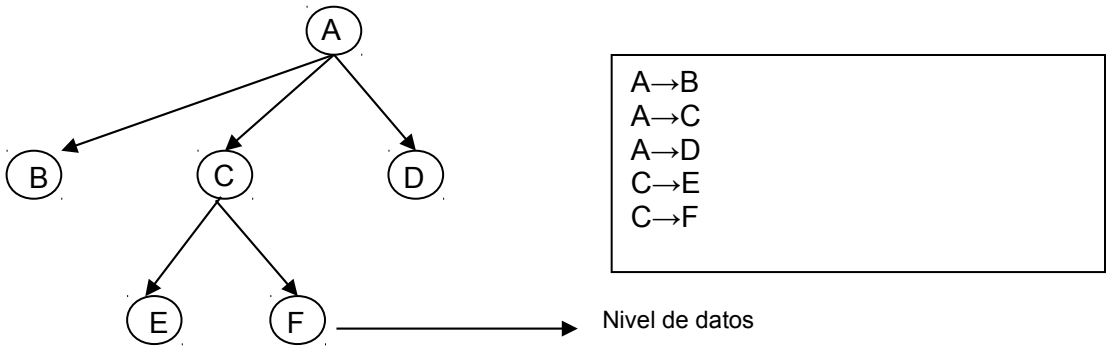
- Niveles de estructura

  Representan los distintos clasificaciones dentro del árbol. Pueden contener también información.

- Niveles de datos

  Son nodos representan múltiples valores, todos asociados a un mismo nivel padre. Pueden ser igualmente considerados como nodos de estructura (en caso de no importarnos los datos). Incluyen un tipo de fuente de datos (inicialmente pensamos en una base relacional) y una clave de acceso a la fuente de datos.

Para implementar la estructura del árbol, usaríamos una tabla en donde se especifica la relación padre hijo:



La estructura completa de la tabla sería:

Relación	Etiqueta	Fuente	Clave de fuente/Valor
AA	Biblioteca		Biblioteca Nacional
AC	Sección		Historia
CE	Encargado		Bernardo
CF	Ejemplares	RT (relacional texto)	1

El nivel F es un nivel de datos. Los datos contenidos en el nivel se almacenarían en una tabla relacional para el tipo de datos texto. La estructura sería la siguiente:

Clave de fuente	Valor
1	Historia Americana
1	Historia Argentina
1	….
1	….
1	Historia Europea

y estará ordenada por los campos "Clave de fuente" y "Valor".

Para acceder a un determinado dato se realizará el siguiente procedimiento:

1) Se accede a la tabla de estructura para localizar el nodo deseado
2) Se verifica si se trata de un nodo de datos o de estructura.
3) Si es un nodo de estructura, se devuelve el valor contenido en la tabla de estructura
4) Si es un nodo de datos
    a. Se comprueba el tipo de fuente de datos y la clave de acceso.
    b. Se accede a la fuente de datos y se consulta por la clave de acceso.
    c. Se devuelve el valor retornado por la fuente de datos.

Como ejemplo:

Consulta: Libros de la sección Historia

1) Se accede al nodo Biblioteca/Sección/Historia/Ejemplares
2) Se comprueba que es un nodo de datos
3) Se establece la fuente de datos, siendo esta relacional texto y la clave de acceso 1
4) Se accede a la tabla relacional de datos texto y se evalúa la clave 1.
5) Se devuelven los valores "Historia Americana" …

Consulta: Encargado sección Historia

1) Se accede al nodo Biblioteca/Sección/Encargado
2) Se comprueba que es un nodo de estructura
3) Se devuelve el valor relacionado con el nodo, o sea, "Bernardo"

Ventajas del modelo

- Se acomoda fácilmente a cambios de estructura

- Permite la inserción, borrado y modificación de datos en los nodos de este tipo.
- Posibilita el acceso a diferentes tipos de información (relacional, textos extensos, directorios, servicios de correo, servicios de búsqueda, etc.) mediante el uso de distintas fuentes de datos.
- Aprovecha la eficiencia de cada fuente de datos.
- Reduce la cantidad de nodos del documento original XML (expresado como grafo o árbol) al comprimir múltiples nodos de datos en un solo nivel de datos.

Descripción de modelo de datos

Almacenaremos los archivos de datos XML (1) en una base de datos relacional. Para almacenar cada uno de los documentos XML utilizaremos el modelo de datos que se describe a continuación.

El modelo de datos consiste en las tablas Estructura, Instancias, Fuentes, y las tablas Fechas, Textos, Numéricos; existirá una tabla por cada posible de tipo de datos que se incluya en el documento XML, los tipos de datos que aceptará XMI estarán preestablecidos. En el caso de ser necesitar indexar documentos XML que contenga un tipo de datos que no esté entre los predefinidos en nuestro modelo, se deberá definir el tipo de dato, generar la tabla de almacenamiento, y desarrollar y/o modificar los procedimientos necesarios para posibilitar su almacenamiento, actualización y consulta de la base, teniendo en cuenta el nuevo tipo de datos.

Almacenamiento de atributos: los atributos del archivo los almacenaremos como elementos del documento, almacenando en la tabla de Estructura el nombre de cada ellos, aclarando que el

tipo de elemento almacenado corresponde a un atributo. El dato de cada uno de los atributos se almacenará en cada una de las tablas de datos que corresponda dependiendo del tipo de dato.

Almacenamiento de tags: los nombres de los tags de cada uno de los documentos XML se almacenaran en la tabla Estructura indicando que el registro corresponde a un tag. Los valores de cada uno de los datos se almacenan en la tabla correspondiente, dependiendo del tipo de dato.

Los atributos y los tags se almacenaran en las mismas tablas. Los nombres de los atributos y los tags se almacenan todos en la tabla de estructura indicando si son atributos o tags. Los valores de los atributos y los tags se almacenan en las mismas tablas de datos, dependiendo el tipo de datos, es decir, los atributos que sean de tipo texto y los elementos de tipo texto son almacenados en la misma tabla.

En el caso de existir ciclos en el documento XML a indexar no representará un problema a la hora de almacenar el documento en la base pues se almacenaran todos los elementos del documento sin tener en cuenta si es un ciclo o no. Por lo tanto DTDs recursivos no afectan la representación del documento en nuestra implementación XMI porque el esquema se toma en base a los datos.

Especificación de las tablas del modelo

El modelo de datos XMI fue implementado en Microsoft SQL Server 7.0, a continuación, la especificación de las tablas que conforman el modelo de datos XMI.

Especificación de la Tabla Estructura

La tabla Estructura almacenará la estructura del documento XML dándole a cada uno de los tags o atributos una clave diferente. Existirá una relación Padre / Hijo entre los niveles de los tags del documento XML. Los atributos serán almacenados como tags indicando en el campo Tipo si corresponde a un tag o a un atributo.

Estructura	Tipo de Dato
Documento [k]	Smallint
Padre	Int
Clave [k]	Int
Tipo	Char
Tag	Varchar(25)
Fuente	Tinyint
Valor	Int

Obs: [k] indica campo perteneciente a la clave.

Se almacenarán en la tabla Estructura no sólo los tags del documento XML sino también los atributos que puedan existir para cada uno de los tags.

La raíz del documento tendrá como Padre su misma Clave. Los tags que no correspondan a la raíz tendrán como Padre a la Clave del tag inmediato superior.

El campo Tipo de la tabla Estructura indicara si el registro corresponde a un tag del documento o a un atributo de un Tag.

El campo Tag contendrá el nombre del Tag o del atributo (según corresponda).

El campo Fuente contiene una la referencia a la tabla donde se almacena el dato.

El campo Valor contendrá la clave de la tabla de almacenamiento de datos referenciada en el campo Fuente. En el caso de ser un elemento con valor, el campo Fuente indicará la tabla donde se almacena el dato y el campo Valor es la Clave Fuente en las tablas de tipos de datos. Si el tag no tiene datos, el campo Valor está vacío como así también el campo Fuente.

La Primary Key de la tabla Estructura está conformada por los campos: Documento y Clave.

La tabla tiene integridad referencial con la tabla Fuentes: y Documentos.

Fuentes.Fuente ← Estructura. Fuente

Documentos.Documento ← Estructura.Documento

Especificación de la Tabla Instancias

La tabla Instancias almacenará por cada tag y cada atributo que contenga el documento XML una nueva instancia. Esta instancia estará relacionada con una clave de la tabla estructura.

También indicará su instancia padre, lo que optimiza las consultas sobre descendientes de una instancia.

En el caso de existir ciclos en el documento cada ocurrencia de un elemento se almacenará como una nueva instancia.

Instancias	Tipo de Dato
Documento [k]	Smallint
Instancia [k]	Int
ClaveEsquema	Int
Padre	Int

La Primary Key de la tabla es: Documento e Instancia.

Integridad de la tabla: Estructura.Clave ←Instancias.ClaveEsquema

Documentos.Documento ← Instancias.Documento

Instancias.Instancia ← Instancias.Padre

Especificación de la tabla Fuentes

La tabla Fuentes almacenará todos los posibles tipos de datos de un documento XML que puedan ser almacenados en las tablas del modelo XMI. Los tipos de datos que están predefinidos, en el caso de ser necesario indexar documentos XML que contengan tipos de datos no predefinidos será necesario modificar el modelo como así también los procesos de carga del documento y consulta y actualización de la base.

Fuentes	Tipo de Dato
Fuente [k]	Tinyint
NombreStoreProc	Varchar (50)

El campo NombreStoreProc tendrá la descripción del tipo de dato. Esta descripción coincidirá con el nombre de una de las tablas de datos (Fechas, Numéricos, Texto, etc.).

La Primary Key de la tabla es el campo Fuente.

Por cada tabla de almacenamiento de datos que se detalla a continuación deberá existir un registro en la tabla Fuentes.

Especificación de las tablas de datos

Tabla Fechas:

La tabla Fuentes almacenará toda información de los tags o atributos que sean de tipo Fecha.

Fechas	Tipo de Dato
Documento [k]	Smallint
CLAVEFUENTE	Int
Clave [k]	Int
Padre	Int
Valor	Datetime

La Primary Key de la tabla está compuesta por el Documento y Clave.

El campo Clave corresponde a la Instancia

El campo Padre corresponde a la Instancia que está inmediatamente arriba de la instancia que almacenamos. La instancia Padre no necesariamente se encuentra almacenada en esta tabla ya que puede ser de otro tipo de dato.

El Valor es el dato concretamente.

Esta tabla tiene las siguientes integridades:

Documentos.Documento ← Fechas.Documento

Estructura.Valor ← Fechas.ClaveFuente

Instancias.Instancia ← Fecha.Clave

Instancias.Instancia ← Fecha.Padre

Tabla Texto:

La tabla Texto almacenará toda información de los tags o atributos que sean de tipo Texto.

Texto	Tipo de Dato
Documento [k]	Smallint
ClaveFuente	Int
Clave [k]	Int
Padre	Int
Valor	Varchar (50)

La Primary Key de la tabla está compuesta por el Documento y Clave.

El campo Clave corresponde a la Instancia

El campo Padre corresponde a la Instancia que está inmediatamente arriba de la instancia que almacenamos. La instancia Padre no necesariamente se encuentra almacenada en esta tabla ya que puede ser de otro tipo de dato.

El Valor es el dato concretamente.

Esta tabla tiene las siguientes integridades:

Documentos.Documento ← Texto.Documento

Estructura.Valor ← Texto.ClaveFuente

Instancias.Instancia ← Texto.Clave

Instancias.Instancia ← Texto.Padre

Tabla Numéricos:

La tabla Numéricos almacenará toda información de los tags o atributos que sean de tipo Numérico.

Numericos	Tipo de Dato
Documento [k]	
ClaveFuente	
Clave [k]	
Padre	
Valor	

La Primary Key de la tabla está compuesta por el Documento y Clave.

El campo Clave corresponde a la Instancia

El campo Padre corresponde a la Instancia que está inmediatamente arriba de la instancia que almacenamos. La instancia Padre no necesariamente se encuentra almacenada en esta tabla ya que puede ser de otro tipo de dato.

El Valor es el dato concretamente.

Esta tabla tiene las siguientes integridades:

Documentos.Documento ← Numéricos.Documento

Estructura.Valor ← Numéricos.ClaveFuente

Instancias.Instancia ← Numéricos.Clave

Instancias.Instancia ← Numéricos.Padre

Tabla IDs:

La tabla IDs almacenará toda información de los tags o atributos que sean de tipo IDs.

Numericos	Tipo de Dato
Documento [k]	
ClaveFuente	
Clave [k]	
Padre	
Valor	

La Primary Key de la tabla está compuesta por el Documento y Clave.

El campo Clave corresponde a la Instancia que define el atributo ID

El campo Padre corresponde a la Instancia que está inmediatamente arriba de la instancia que almacenamos. La instancia Padre no necesariamente se encuentra almacenada en esta tabla ya que puede ser de otro tipo de dato.

El Valor es el dato concretamente, en este caso el valor dado al ID, que puede ser referenciado en cualquier instancia del documento a través de un atributo IDREF.

Esta tabla tiene las siguientes integridades:

Documentos.Documento ← Ids.Documento

Estructura.Valor ← IDs.ClaveFuente

Instancias.Instancia ← IDs.Clave

Instancias.Instancia ← IDs.Padre

Especificación de la tabla Documentos

El modelo permite almacenar más de un documento. La tabla Documentos almacena el nombre de cada uno de los documentos XML que son almacenados en el modelo XMI. A cada documento se le asigna una clave que relacionará en el resto de las tablas del modelo los registros con cada uno de los documentos.

Documentos	Tipo de Dato
Documento [k]	Smallint
Nombre	Varchar

Ejemplo de representación en XMI

A continuación se muestra un documento XML y su representación en el modelo XMI:

Documento XML original

```
<?xml version="1.0" ?>
  - <!--
    <?xml-stylesheet href="XSL\JavaXML.html.xsl" type="text/xsl"?>
    <?xml-stylesheet href="XSL\JavaXML.wml.xsl" type="text/xsl"
                     media="wap"?>
    <?cocoon-process type="xslt"?>
    <!DOCTYPE JavaXML:Book SYSTEM "DTD\JavaXML.dtd">
  -->
 - <!--
  Java and XML
  -->
- <JavaXML:Book xmlns:JavaXML="http://www.oreilly.com/catalog/javaxml/">
  <JavaXML:Title>Java and XML</JavaXML:Title>
- <JavaXML:Contents>
- <JavaXML:Chapter focus="XML">
  <JavaXML:Heading>Introduction</JavaXML:Heading>
  <JavaXML:Topic subSections="7">What Is It?</JavaXML:Topic>
  <JavaXML:Topic subSections="3">How Do I Use It?</JavaXML:Topic>
  <JavaXML:Topic subSections="4">Why Should I Use It?</JavaXML:Topic>
```

<JavaXML:Topic subSections="0">**What's Next?**</JavaXML:Topic>

</JavaXML:Chapter>

- <JavaXML:Chapter focus="**XML**">

<JavaXML:Heading>**Creating XML**</JavaXML:Heading>

<JavaXML:Topic subSections="0">**An XML Document**</JavaXML:Topic>

<JavaXML:Topic subSections="2">**The Header**</JavaXML:Topic>

<JavaXML:Topic subSections="6">**The Content**</JavaXML:Topic>

<JavaXML:Topic subSections="1">**What's Next?**</JavaXML:Topic>

</JavaXML:Chapter>

- <JavaXML:Chapter focus="**Java**">

<JavaXML:Heading>**Parsing XML**</JavaXML:Heading>

<JavaXML:Topic subSections="3">**Getting Prepared**</JavaXML:Topic>

<JavaXML:Topic subSections="3">**SAX Readers**</JavaXML:Topic>

<JavaXML:Topic subSections="9">**Content Handlers**</JavaXML:Topic>

<JavaXML:Topic subSections="4">**Error Handlers**</JavaXML:Topic>

<JavaXML:Topic subSections="0">**A Better Way to Load a Parser**</JavaXML:Topic>

<JavaXML:Topic subSections="4">**"Gotcha!"**</JavaXML:Topic>

<JavaXML:Topic subSections="0">**What's Next?**</JavaXML:Topic>

</JavaXML:Chapter>

<JavaXML:SectionBreak />

- <JavaXML:Chapter focus="**Java**">

<JavaXML:Heading>**Web Publishing Frameworks**</JavaXML:Heading>

<JavaXML:Topic subSections="4">**Selecting a Framework**</JavaXML:Topic>

<JavaXML:Topic subSections="4">**Installation**</JavaXML:Topic>

<JavaXML:Topic subSections="3">**Using a Publishing Framework**</JavaXML:Topic>

<JavaXML:Topic subSections="2">**XSP**</JavaXML:Topic>

<JavaXML:Topic subSections="3">**Cocoon 2.0 and Beyond**</JavaXML:Topic>

<JavaXML:Topic subSections="0">**What's Next?**</JavaXML:Topic>

</JavaXML:Chapter>

</JavaXML:Contents>

- <JavaXML:References>

- <JavaXML:Reference>

<JavaXML:Name>**The W3C**</JavaXML:Name>

<JavaXML:Url>**http://www.w3.org/Style/XSL**</JavaXML:Url>

</JavaXML:Reference>

- <JavaXML:Reference>

<JavaXML:Name>**XSL List**</JavaXML:Name>

<JavaXML:Url>**http://www.mulberrytech.com/xsl/xsl-list**</JavaXML:Url>

</JavaXML:Reference>

</JavaXML:References>

- <JavaXML:Copyright>

-

- <table cellpadding="**0**" cellspacing="**1**" border="**1**" bgcolor="**Black**">

- <tr>

- <td align="**center**">

- <table bgcolor="**White**" border="**2**">

- <tr>

- <td>

Copyright O'Reilly and Associates, 2000

</td>

```
</tr>
</table>
</td>
</tr>
</table>
</center>
</JavaXML:Copyright>
- <!--
<JavaXML:Copyright>&OReillyCopyright;</JavaXML:Copyright>
--> 
</JavaXML:Book>
```

Representación XMl

Tabla Documentos

Documento	Nombre
1	Indice.xml

Tabla Fuentes

Fuente	NombreStoreProc
Texto	1
Numéricos	2
Fecha	3

Tabla Estructura

Documento	Padre	Clave	Tipo	Tag	Fuente	Valor
1	1	1	T	JavaXML:Book		
1	1	2	A	xmlns:JavaXML	1	1
1	1	3	T	JavaXML:Title	1	2
1	1	4	T	JavaXML:Contents		
1	1	5	T	JavaXML:References		
1	1	6	T	JavaXML:Copyright		
1	4	7	T	JavaXML:Chapter		
1	7	8	A	Focus	1	3
1	4	9	T	JavaXML:SectionBreak		
1	5	10	T	JavaXML:Reference		
1	6	11	T	Center		
1	11	12	T	Table		
1	12	13	A	Cellpadding	2	1
1	12	14	A	Cellspacing	2	2
1	12	15	A	Border	2	3
1	12	16	A	Bgcolor	1	4
1	12	17	T	Tr		
1	17	18	T	Td		
1	18	19	A	Align	1	5
1	18	20	T	Table		
1	20	21	A	Bgcolor	1	6
1	20	22	A	Border	2	4
1	20	23	T	Tr		
1	23	24	T	Td		
1	24	25	T	Font	1	7
1	25	26	A	Size	2	5
1	7	27	T	JavaXML:Heading	1	8
1	7	28	T	JavaXML:Topic	1	9
1	28	29	A	SubSections	2	6
1	10	30	T	JavaXML:Name	1	10
1	10	31	T	JavaXML:Url	1	11

Tabla instancias

Documento	Instancia	ClaveEsquema	Documento	Instancia	ClaveEsquema
1	1	1	1	47	27
1	2	2	1	48	28
1	3	3	1	49	29
1	4	4	1	50	28
1	5	7	1	51	29
1	6	8	1	52	28
1	7	27	1	53	29
1	8	28	1	54	28
1	9	29	1	55	29
1	10	28	1	56	28
1	11	29	1	57	29
1	12	28	1	58	28
1	13	29	1	59	29
1	14	28	1	60	5
1	15	29	1	61	10
1	16	7	1	62	30
1	17	8	1	63	31
1	18	27	1	64	10
1	19	28	1	65	30
1	20	29	1	66	31
1	21	28	1	67	6
1	22	29	1	68	11
1	23	28	1	69	12
1	24	29	1	70	13
1	25	28	1	71	14
1	26	29	1	72	15
1	27	7	1	73	16
1	28	8	1	74	17
1	29	27	1	75	18
1	30	28	1	76	19
1	31	29	1	77	20
1	32	28	1	78	21
1	33	29	1	79	22
1	34	28	1	80	23
1	35	29	1	81	24
1	36	28	1	82	25
1	37	29	1	83	26
1	38	28			
1	39	29			
1	40	28			
1	41	29			
1	42	28			
1	43	29			
1	44	9			

1	45	7		
1	46	8		

Tabla Fechas

Documento	ClaveFuente	Clave	Padre	Valor

Tabla Numéricos

Documento	ClaveFuente	Clave	Padre	Valor
1	6	9	8	7
1	6	11	10	3
1	6	13	12	4
1	6	15	14	0
1	6	20	19	0
1	6	22	21	2
1	6	24	23	6
1	6	26	25	1
1	6	31	30	3
1	6	33	32	3
1	6	35	34	9
1	6	37	36	4
1	6	39	38	0
1	6	41	40	4
1	6	43	42	0
1	6	49	48	4
1	6	51	50	4
1	6	53	52	3
1	6	55	54	2
1	6	57	56	3
1	6	59	58	0
1	1	70	69	0
1	2	71	69	1
1	3	72	69	1
1	4	79	77	2
1	5	83	82	-1

Tabla Texto

Documento	ClaveFuente	Clave	Padre	Valor
1	1	2		1http://www.oreilly.com.catalog/javaxml
1	2	3		1Java and XML
1	3	6		5XML
1	8	7		5Introduction
1	9	8		7What Is It?
1	9	10		7How Do I Use It?
1	9	12		7Why Should I Use It?
1	9	14		7What's Next?
1	3	17		16XML
1	8	18		16Creating XML
1	9	19		16An XML Document
1	9	21		16The Header
1	9	23		16The Content
1	9	25		16What's Next?
1	3	28		27Java
1	8	29		27Parsing XML
1	9	30		27Getting Prepared
1	9	32		27SAX Readers
1	8	34		27Content Handlers
1	9	36		27Error Handlers
1	9	38		27A Better Way to Load a Parser
1	9	40		27"Gotcha!"
1	9	42		27What's Next?
1	3	46		45Java
1	8	47		45Web Publishing Frameworks
1	9	48		45Selecting a Framework
1	9	50		45Installation
1	9	52		45Using a Publishing Framework
1	9	54		45XSP
1	9	56		45Cocoon 2,0 and Beyond
1	9	58		45What's Next?
1	10	62		61The W3C
1	11	63		61http://www.w3.org/Style/XSL
1	10	65		64XSL List
1	11	66		64http://www.mulberrytech.com/xsl/xsl-list
1	4	73		69Black
1	5	76		75center
1	6	78		77White
1	7	82		81Copyright O'Reilly and Associates, 2000

Equivalencia entre un documento XML y su almacenamiento usando XMI

Representación de un documento XML

Representaremos al documento XML como el grafo unidireccional G = <N, T, A, D, RTT, RTA, RAD, RTD, RTDC >

Donde

N: nombre de documento

T: conjunto de Tags (tag t)

A: conjunto de Atributos (atributo a)

D: conjunto de Datos (dato = d), donde cada dato puede corresponder a tipos diferentes, por ejemplo: texto, numérico, fecha.

RTT: Relaciones entre Tags (t_t)

RTA: Relaciones entre Tags y Atributos (t_a)

RAD: Relaciones entre Atributos y Datos (a_d)

RTD: Relaciones entre Tags y Datos (t_d)

RTDC: Relaciones entre r $\in$ RTT o r $\in$ RTD y Documento y relaciones entre r $\in$ RTA o r $\in$ RAD y Documento.

Las relaciones RTT estarán conformadas por una dupla (t1, t2) donde t1, t2 son tags del documento y t1 tiene a t2 como primer nivel de tag anidado.

Las relaciones RTA estarán conformadas por una dupla (t1, a1) donde t1 es tag del documento y a1 es atributo de t1.

Las relaciones RAD estarán conformadas por una dupla (a1, d1) donde a1 atributo del documento y d1 es el valor de a1.

Las relaciones RTD estarán conformadas por una dupla (t1, d1) donde t1 es tag del documento y d1 es el valor de t1.

Las relaciones RTDC estarán conformadas por una dupla (t1, dc1) donde t1 es tag del documento y d1 es el documento XML o dupla (a1, dc1) donde a1 es atributo del documento y d1 es el documento XML.

Existirá un único tag (t1) en G / (t2,t1) $\notin$ RTT, con t1,t2 $\in$ T. Este tag t1 corresponderá al tag raíz del documento.

$\forall$ rtt $\in$ RTT, con rtt = (t1,t2), t1,t2 $\in$ T.

Si rtt = (t1,t2), t1,t2 $\in$ T ➔ no Existe rtd $\in$ RTD / rtd = (t1,d1) donde d1 es un dato.

Representaremos el documento XML a través de una estructura de grafo unidireccional, donde el único nodo que no tiene padre corresponde al tag raíz del documento.

Si existe rtt=(a,b) $\in$ RTT o rta(a,b) $\in$ RTA ➔ a$\in$ T , es decir, si un nodo tiene por lo menos otro tag o atributo anidado, el nodo corrresponde a un tag del documento.

Los nodos sin hijos pueden representar tags (sin otros tags anidados), atributos o IDRef/IDRefs.

Cabe aclarar que los IDRefs tendrán un almacenamiento en XMI similar a los atributos, con la excepción de que existirá una tabla especial para cargar estas referencias.

Los nodos sin hijos pueden contener datos, que se almacenaran en la tabla datos de XMI que corresponda.

Para cada una de las relaciones r / (r $\in$ RTT $\vee$ r$\in$ RTA $\vee$ r$\in$ RAD $\vee$ r$\in$ RTD) existirá una única relación r1 $\in$ RTDC, con r1= (d1,r2) donde (r2$\in$ RTT o r2$\in$ RTA $\vee$ r2$\in$ RTD $\vee$ r2$\in$ RAD)

Almacenamiento del documento XML en XMI

Cada una de las duplas r1= (d1,r2) que pertenecen a RTDC, con r2 (a,b), se corresponde a un único registro de la tabla Estructura del modelo XMI, donde el documento de la tabla Estructura se corresponde con la clave de d1, a se corresponde con el campo Padre de la tabla Estructura y b se corresponde con el campo Clave de la tabla Estructura.

Cada una de estas duplas pertenecientes a RTDC puede contener más de un dato en el documento. Las relaciones entre las duplas RTDC y los datos propiamente dichos se denominarán Instancias (I).

Cada uno de las instancias de atributos y tags que aparezcan en el documento se corresponderá con una nueva Instancia que se relacionará en la tabla Instancias con la clave de la tabla Estructura y la clave del documento XML. Cada una de las instancias tendrá una clave única en XMI.

Cada uno de los datos de tags o de atributos se relacionara unívocamente con una instancia de XMI.

Cada una de esta instancia será única en el documento y por lo tanto también será única en el modelo XMI, por estar relacionada con solo una dupla RTDC.

Las relaciones entre las instancias, y las claves del atributo o tag y el documento se almacenará en la tabla de Instancia.

El dato de la Instancia se almacenará en la tabla de datos que corresponda.

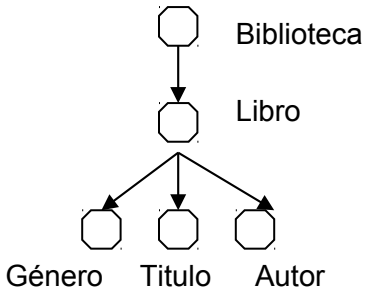
Se le asignará en el modelo XMI una clave única al documento a almacenar. De esta forma se podrá reestablecer a través de consultas al modelo la estructura original del documento XML y las instancias que la componen ya que todas las relaciones de la estructura entre tags y atributos como así también las instancias estarán relacionadas al documento origen.

Ejemplo

Documento Bilioteca.xml

```
<Biblioteca>
        <Libro Genero: Novela>
                <Titulo>Ficciones</Titulo>
                <Autor>Borges</Autor>
        </Libro>
        <Libro>
                <Titulo>La invención de Morel</Titulo>
                <Autor>Bioy Casares</Autor>
        </Libro>
        <Libro>
                <Titulo>Anaconda</Titulo>
                <Autor>Horacio Quiroga</Autor>
        </Libro>
</Biblioteca>
```

Estructura del grafo representa al documento Biblioteca.xml



Las relaciones son:

RTT = {(Biblioteca, Libro), (Libro,Titulo), (Libro, Autor)}

RTA = {(Libro, Genero)}

RTD = {(Titulo, Ficciones), (Titulo, La invencion de Morel), (Titulo, Anaconda), (Autor, Borges), (Autor, Bioy Casares), (Autor, Horacio Quiroga)}

RAD = {(Genero, Novela)}

RTDC = {(Biblioteca, Biblioteca), (Libro, Biblioteca), (Titulo, Biblioteca), (Autor, Biblioteca), (Genero, Biblioteca)}

Cada tag y atributo del documento XML tienen asignado una clave única (CT, CA respectivamente) en la tabla Estructura.

Todo Tag t, Atributo A estará relacionado un nodo padre, que siempre corresponderá a un Tag, pues los Atributos no pueden tener hijos (según estructura de árbol). Cada una de estas relaciones estará relacionada con un único Documento (clave CD) en XMI, ➔ existirá una única relación en XMI para cada RTDC

El documento se almacenará en el modelo XMI y se definirá para el mismo una clave única (CD).

Las relaciones que existen en un documento XML se cargarán en la tabla Estructura. Cada una de estas relaciones estará relacionado con el documento origen xml (CD).

Los datos de tags y atributos se almacenaran en forma similar con la única característica que en la tabla Estructura se indicará si la clave corresponde a un tag o a un atributo (campo Tipo de la tabla Estructura). Los datos tienen definido un único tipo de dato para la carga del mismo en XMI, el valor se almacena en una única tabla de datos. El dato no se carga en más de una tabla con la misma clave.

Cada instancia de un tag/atributo de un único documento se define a partir de una instancia. La instancia es única en el documento y se relaciona con una única clave tag/atributo.

Es importante aclarar que la equivalencia de XML y XMI, no se considera orden entre sus elementos. La equivalencia considera que se mantengan las relaciones entre tags, atributos y documentos y los datos que contenga cada uno de los elementos y que pueda ser reconstruido el documento original XML a partir de las consultas disponibles.

Mantenimiento del orden dentro del documento

El orden de la información presente en el documento XML original es mantenido al momento de almacenarlo en el modelo XMI. Esto se logra a partir de la utilización de la tabla de instancias, la cual almacena un registro por cada tag presente dentro del documento XML. Estos registros poseen como clave un valor autoincremental denominado "instancia", que determina en que orden aparece el tag dentro del documento, para un recorrido del tipo BFS del árbol que lo representa.

En todas las primitivas de acceso a datos se ordena el resultado de acuerdo a la "instancia" de cada tag, por lo tanto el orden se encuentra asegurado. El tema es de fundamental importancia por ejemplo en operaciones como la reconstrucción completa del documento original, en donde el orden de los datos debe ser preservado.

Cabe mencionar que el orden en que los datos son extraídos del documento original corresponde al orden en que los mismos son reconocidos y representados dentro del modelo DOM.

Decisiones de diseño

Tablas individuales por tipo de datos

El modelo XMI utiliza para el almacenamiento de datos una tabla individual para cada uno de los tipos de datos que se presentan en el documento XML original.

Los tipos de datos incluidos inicialmente en nuestra implementación son los siguientes:

- Numérico
- Texto
- Fecha

A partir de extensiones en la implementación, podrán incorporarse nuevos tipos de datos, desde algunos simples como el tipo de datos moneda a otros sumamente complejos, como por ejemplo pociones de audio o video digital.

Se eligió la implementación de tablas individuales como alternativa al almacenamiento de toda la información en formato carácter (texto) por considerar se más eficiente la administración de los datos de acuerdo al tipo de los mismos.

Esta mayor eficiencia se hace notoria en los siguientes aspectos:

- Mejor utilización del espacio de almacenamiento.

  Tomando como ejemplo un valor numérico, como el 112120,53695, se necesitan 12 bytes para representarlo como texto y solo 4 bytes para representarlo como un valor numérico (tipo float). Este ahorro de espacio redunda en una mayor velocidad en la recuperación de los datos y un consumo menor de recursos de almacenamiento. Mas aún teniendo en cuenta el modelo de almacenamiento relacional elegido y la utilización extensiva de índices que se hace dentro del modelo XMI.

- Simplicidad en las comparaciones y en el ordenamiento.

  Se pueden producir inconsistencias en las comparaciones lógicas entre valores numéricos representados como texto, que deberían tratarse de forma cuidadosa de elegirse dicha representación. En caso de que los valores numéricos se almacenen en un tipo especializado estas inconsistencias desaparecen. Se pone como ejemplo el valor 321 , almacenado como texto. Dada la comparación de “321”>”123”, siendo “123” otro valor numérico representado (el entero 123), el valor de la comparación se evaluará como Verdadero, siendo esto incorrecto si aplicamos la comparación a los valores numéricos originales.
  Un caso similar se presenta en el ordenamiento, donde si no se toman medidas correctivas, los números 123,5 y 32 aparecerían incorrectamente ordenados si se representan como texto y correctamente ordenados si se representan con tipos numéricos

- Utilización de funciones específicas

  Si los datos son almacenados relacionalmente en campos del tipo adecuado, será posible la utilización sobre ellos de funciones específicamente diseñadas para estos tipos de datos que implementan un procesamiento más complejo y apropiado a las funciones disponibles para los campos de texto. Ejemplos de estas funciones son:

- Para los campos numéricos: Mod, Int, Sqr, Abs.
- Para los campos de texto: Like, Substring, Left, Right,UpperCase.
- Para los campos de fecha: Year, Month, Day, DateDiff, DateAdd.

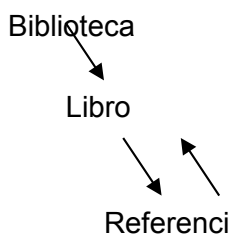
Generación de metadatos

El modelo XMI prevé la generación de información de metadatos a partir de los mismos datos contenidos en el documento XML.

Fue elegido este procedimiento en vez de utilizar un esquema de datos definido anteriormente (como un DTD o XML Schema) para facilitar la inclusión al modelo de información no estructurada que no cuente inicialmente con un esquema de datos adjunto.

Otro motivo por el cual se elige procesar los datos y de ellos extraer el esquema es que de esta manera se elimina cualquier recursividad presente en el esquema de datos. El tema de la recursividad podría volverse sumamente difícil de manejar si la misma no es eliminada en las primeras instancias de procesamiento.

Para ejemplificar lo explicado, tomemos el siguiente esquema datos:



Podemos apreciar un ciclo en en grafo que representa el esquema de datos, mostrando una relación recursiva entre un libro y sus referencias

Mediante el procesamiento inicial de los datos que realiza el modelo XMI, la recursividad presentada se “aplana”, creando un esquema de datos en donde el tag “Referencias” no se une recursivamente con otro tag “Libro”, sino que tiene un tag hijo de ese tipo.

Las propiedades de eliminación de la recursividad y de independización de un esquema de datos hacen más flexible al modelo XMI, a costo de un procesamiento incial de los datos. Este procesamiento inicial es utilizado igualmente para generar diferentes valores necesarios para el posterior almacenamiento de los datos en una base relacional.

Implementación de esquemas XML-Schema

Por una rezón de completitud y de inclusión de tecnologías y estándares el modelo XMI soporta también la utilización de esquemas de metadatos que se adecuen al standard XML-Schema, para documentos que así lo provean.

En el caso de disponer del esquema mencionado, el proceso inicial de parsing, en donde se extrae la estructura del documento y se infieren los tipos del mismo, ya no es necesaria. Esta

etapa se reemplaza por un módulo que convierte la estructura XML-Schema a las estructuras de datos del modelo XMI.

Limites en el tamaño de los documentos

En el momento de diseñar la base de datos relacional que da soporte al almacenamiento de nuestro modelo fue necesario poner especial atención el los tipos de datos de ciertos campos de las tablas de la base. El tipo de datos elegido para estos campos determinará de manera precisa la cantidad de información que la implementación del modelo puede almacenar.

Los campos más relevantes en la estructura relacional de almacenamiento son:

Tablas de estructura

Clave (Tabla Estructura)

Padre (Tabla Estructura)

TipoFuente (Tabla Estructura)

ClaveFuente (Tabla Estructura)

Instancia (Tabla Instancias)

Tablas de datos

Clave (cada tabla de datos)

El campo Clave de la tabla Estructura limita la cantidad máxima de tags distintos que el modelo XMI puede almacenar (incluyendo todos los documentos almacenados en la base de datos). El campo Padre es una referencia al campo Clave de la misma tabla, por lo que se mantiene el límite impuesto anteriormente.

El campo ClaveFuente de la tabla estructura indica, para cada TipoFuente, un valor autoincremental con la cantidad de tags en la base de datos del tipo indicado por TipoFuente. Por lo tanto no podremos almacenar más tags que el rango máximo del tipo de datos usado para representar el valor “ClaveFuente” de cada uno de los tipos de datos definidos por “TipoFuente”

Proceso de parsing

Al comenzar el proceso de parsing contamos con un documento XML sintácticamente correcto, que cumple con la especificación World Wide Web Consortium (W3C) Document Object Model (DOM) Niveles 1 y 2. A partir de este documento crearemos una representación en nuestro modelo de almacenamiento y consulta de los datos contenidos en el mismo.

El proceso completo de parsing o reconocimiento se compone de las siguientes etapas:

1. Inicialmente se inserta un registro en la tabla Documentos obteniéndose como resultado de la operación un identificador de documento, cuya referencia será usada posteriormente en todas las tablas del modelo XMI.

2. Se carga el documento original en una estructura de datos que corresponde a la implementación de un objeto documento en el DOM. Esto nos permite validar la correctitud sintáctica del documento y facilitar su reconocimiento y manejo.

3. Una vez obtenida la representación del documento en el modelo DOM, se procede a recorrer esta representación, analizando particularmente los tipos de nodo Atributo y Elemento (especificación NodeType DOM). Se recorren los nodos del documento expresado dentro de DOM con un algoritmo del tipo BFS (Breadth First Search).

    a. Para cada uno de estos nodos encontrados en el documento se analizan los atributos pertenecientes al mismo, junto al valor de tipo texto que lo pueda acompañar (texto de un nodo elemento).

    b. Cada uno de los nodos recorridos es insertado en una estructura de datos denominada “Árbol comprimido”, en donde se representa la estructura de tags y atributos del documento original, vista a partir de la representación de este en el modelo DOM. El proceso de generación del árbol comprimido es detallado extensivamente mas adelante en este documento.

4. El árbol comprimido resultante de la etapa anterior consta en cada uno de sus nodos de la siguiente información:

    - Clave del nodo.
    - Clave del nodo padre.
    - Tag o nombre de atributo.
    - Tipo de dato del tag o del atributo.
    - Tipo de elemento (tag o atributo).

5. Para facilitar la manipulación de los datos que contiene el árbol comprimido generado en el paso anterior, transformamos esta estructura de datos en otra provista por el ambiente de implementación, semejante a una tabla relacional.

6. Se permite vía interfase del usuario la modificación de los tipos de datos inferidos de los tags y atributos del documento original. Los tipos de datos solo pueden ser modificados por tipos compatibles a los obtenidos inicialmente, por ejemplo se puede modificar un tipo Fecha a uno Texto, pero no un tipo Texto a uno Numérico.

7. A partir de la estructura de datos disponible se crea ahora la tabla de estructura del modelo XMI descripta en el título anterior. Para ello se recorre el árbol comprimido y se almacena cada nodo del mismo en una tabla relacional de nuestra base de datos XMI.

8. Ya prácticamente definida la tabla de estructura del modelo XMI es ahora necesaria la incorporación de los datos del documento dentro del almacenamiento XMI. Como primer paso para ello, por cada atributo o tag distinto (en el path donde se encuentra) se actualiza un campo en la tabla estructura del modelo representando la clave que este tag o atributo tendrá dentro de la tabla de datos específica que almacenará su información

   Además de esto se deben crear entradas en diversas tablas como:

   - Instancias

     Se almacena una nueva instancia por cada tag o atributo presente en el documento original.

- Tablas particulares de cada tipo de datos

  Se guarda la información contenida en cada tag o atributo del documento original de acuerdo al tipo de dato definido para ese tag o atributo en la tabla de estructura del modelo XMI y a la clave que el tag o atributo tenga en esta misma tabla.

Creación y uso del árbol comprimido

El árbol comprimido será utilizado al momento de reconocer el documento XML original para detectar en él los tags y atributos que contiene. La estructura de árbol comprimido corresponde a un árbol del tipo Trie[i]

En una primera instancia, se recorre el documento XML origen en forma profunda (DFS) y para cada uno de los nodos visitados (tanto sean tags como atributos) se ejecuta una llamada a la interfase de agregado de nodos del árbol comprimido. Esta llamada intenta agregar al árbol comprimido un nuevo nodo, que representa un camino posible desde la raíz del documento XML original (representado en el modelo DOM) hasta una de sus hojas o nodos internos.

Por ejemplo si el documento fuese:

```
<Biblioteca>
        <Libro Genero="Fantasia">
                <Publicacion>05/06/2002</Publicacion>
                <Autor>Borges</Autor>
```

```
            <FechaNac>04/05/1905</FechaNac>
            <Nacionalidad>Argentino</Nacionalidad>
            <Sexo>M</Sexo>
            <Titulo>Ficciones</Titulo>
            <CantEjemplares>10</CantEjemplares>
        </Libro>
</Biblioteca>
```

Se agregarán al árbol comprimido los siguientes nodos:

Biblioteca

Biblioteca/Libro

Biblioteca/Libro/Genero

Biblioteca/Libro/Publicacion

Biblioteca/Libro/Autor

Biblioteca/Libro/FechaNac

Biblioteca/Libro/Nacionalidad

Biblioteca/Libro/Sexo

Biblioteca/Libro/Titulo

Biblioteca/Libro/CantEjemplares

El ejemplo anterior nos muestra la representación de un documento que solo contiene un valor por cada uno de los tags que lo conforman. Si el documento original fueran de la forma:

```
<Biblioteca>
        <Libro>
```

```
        <Titulo>Ficciones</Titulo>
        <Autor>Borges</Autor>
    </Libro>
    <Libro>
        <Titulo>La invención de Morel</Titulo>
        <Autor>Bioy Casares</Autor>
    </Libro>
    <Libro>
        <Titulo>Anaconda</Titulo>
        <Autor>Horacio Quiroga</Autor>
    </Libro>
</Biblioteca>
```

El árbol comprimido resultante sería:

Biblioteca

Biblioteca/Libro

Biblioteca/Libro/Titulo

Biblioteca/Libro/Autor

Aunque exista más de un elemento en el documento XML original con idéntico camino desde la raíz.

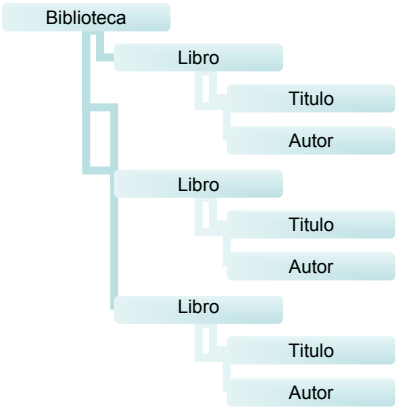
Es aquí donde se puede apreciar la utilidad del árbol comprimido. La función de incorporación de nodos chequea si el camino ya está representado en el árbol y de ser así no continua con el

ingreso. Esto nos permite obtener una representación de los tags y atributos presentes en el documento sin repeticiones, sumamente útil en el proceso de reconocimiento y almacenamiento. También esta representación será usada en la etapa de consulta al motor XMI para la validación de paths.

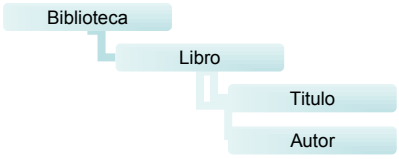
Gráficamente apreciamos a continuación la representación de la estructura de un documento XML usando un árbol comprimido:

```
<Biblioteca>
        <Libro>
                <Titulo>Ficciones</Titulo>
                <Autor>Borges</Autor>
        </Libro>
        <Libro>
                <Titulo>La invención de
Morel</Titulo>
                <Autor>Bioy Casares</Autor>
        </Libro>
        <Libro>
                <Titulo>Anaconda</Titulo>
                <Autor>Horacio Quiroga</Autor>
        </Libro>
</Biblioteca>
```

Documento XML orignal



Representación de los elementos documento en el modelo DOM



Árbol comprimido resultante

En el proceso de agregado de nodos al árbol comprimido se verifica el tipo de datos del nodo a agregar. Si el path del nodo a agregar ya estaba representado en el árbol comprimido, se actualiza el tipo de datos del nodo ya existente en el árbol comprimido. Si el path no aparece representado, el nuevo nodo se agrega indicando un tipo de datos “desconocido”. En la próxima sección se detalla la función que permite definir el tipo de datos que se aplica en el caso de actualizaciones.

Definición de los tipos de datos

La estructura de árbol comprimido que anteriormente se describe contiene cierta información en cada uno de sus nodos, entre la que se encuentra el tipo de datos que corresponde al valor del mismo.

Para determinar el tipo de dato efectivo que se aplica a un nodo del árbol comprimido y por lo que éste representa a un tag del documento XML original, se utiliza una función cuya definición y tabla de valores se especifica a continuación:

Función que determina el tipo de dato efectivo

TipoDatoEfectivo (x,y)

TipoDatoEfectivo: TipoDato x TipoDato $\rightarrow$ TipoDato

Tabla de valores de la función

Tipos dato nodo(x) / Tipo dato árbol (y)	Fecha	Numérico	Texto	Definido
Fecha	Fecha	Texto	Texto	a definir
Numérico	Texto	Numérico	Texto	a definir
Texto	Texto	Texto	Texto	a definir
Definido	a definir	a definir	a definir	a definir

Siendo el parámetro x el tipo de dato del nodo a evaluar y el parámetro y el tipo de dato ya existente en el nodo del árbol comprimido.

El tipo de datos “definido” corresponde a tipos de datos definidos por el usuario. En el caso de estos tipos de datos, el usuario deberá suministrar una función bien definida que reciba como parámetro un valor y devuelva como resultado si este valor pertenece o no al tipo de datos del usuario.

Modificación del tipo de datos inferido

El usuario podrá modificar el tipo de dato inferido a partir del documento inicial y asignarle un tipo de dato distinto, siempre y cuando el tipo asignado por el usuario sea compatible con el tipo de datos previo, por ejemplo no se puede pasar un tipo de dato texto a un tipo de dato numérico.

Comparación entre modelos

Diferencias y similitudes entre XMI y el modelo “Edge”

En la presente sección se compara el modelo presentado en este trabajo de tesis con otro, denominado “edge”[ii]. Ambos modelos se tratan de implementaciones del almacenamiento de documentos XML en bases de datos relacionales.

El modelo de almacenamiento consiste en almacenar un documento XML como un conjunto de pares o ejes, indicando nodo origen y destino. Se utiliza un grafo dirigido como abstracción al documento XML. Existe una tabla que almacena estas relaciones, indicando además el tag correspondiente y el valor del mismo en caso de corresponder.

Comparando este modelo con el modelo XMI surgen inmediatamente varias diferencias a saber:

Estructura

- El concepto y funcionalidad de tabla de instancias utilizada por el modelo XMI se encuentra embebido dentro de la tabla de ejes. La tabla instancia permite reconstruir las relaciones de padre e hijo entre las múltiples ocurrencias de un mismo tag al final de un mismo camino. Estas relaciones se encuentran en el modelo Edge indicadas a partir de los campos Origen y Destino.

- El modelo Edge presenta tantas entradas en la tabla de ejes como tags existan en el documento, sin importar si estos se repiten o no. En el modelo XMI se utiliza la tabla Estructura, en donde solo aparecerá una vez cada tag que pertenezca a un mismo camino dentro del documento.

Almacenamiento

- El modelo Edge repite información para cada uno de los ejes correspondientes a un mismo tag, por ejemplo el campo Name. En cambio la tabla Estructura del modelo XMI evita este problema.

- La información contenida en la tabla de ejes del modelo Edge está desnormalizada, trayendo esto aparejado serios problemas de actualización, modificación y borrado propios de estos esquemas de datos. La descomposición en las tablas Estructura, Instancias y las tablas de datos del modelo XMI corresponden a un esquema normalizado de datos. La combinación de las mismas genera el esquema de datos original sin registros espurios. Este esquema normalizado es especialmente conveniente para soportar modificaciones posteriores al proceso de parsing.

Consultas

- Para acceder a todos los datos de un determinado camino dentro del documento, el modelo Edge debe realizar una gran cantidad de accesos, ya que los posibles caminos dentro del documento solo se encuentran indicados en las relaciones entre los nodos de datos (entradas a la tabla de ejes). El modelo XMI permite un acceso rápido y directo para el tipo de consulta mencionada a partir de un proceso sobre la tabla Estructura.

- En el modelo Edge, instancias de un tag dentro de un mismo camino en el documento pueden encontrarse dispersas por todo el documento mientras que en el modelo XMI las mismas se encuentran contiguas dentro de la tabla de datos correspondiente (de acuerdo al tipo de datos del valor del tag).

A partir de las cuestiones antes mencionadas llegamos a una serie de conclusiones. La primera de ellas y una de las más importantes es que si bien el modelo XMI no se inspiró en ningún otro modelo de almacenamiento de datos XMI en esquemas relacionales como el Edge, el mismo puede parecer una mejora o adaptación de este. Esta mejora se aprecia en el sentido de que almacenar los datos de acuerdo a un esquema normalizado provee innumerables ventajas al momento de modificar el contenido del documento XML y reflejar esas modificaciones en el almacenamiento relacional. Pocos de los modelos actualmente propuestos ofrecen mecanismos que contemplen esta cuestión, mientras que el modelo XMI es altamente flexible a los cambios de estructura y datos del documento XML original.

Otra conclusión interesante es que el modelo XMI parece llegar a equilibrar los beneficios de los distintos tipos de acceso a los datos, haciendo más eficientes tanto los accesos puntuales a una

cantidad pequeña de tags y datos como las consultas extensas que devuelven una importante porción del documento.

Comparación con Path

Consulta de Instancias

PathTox: se alcanza la instancia directamente accediendo a la tabla con mismo tag. Tomar el valor de igual forma. Simple implementación de condiciones sobre los valores ya que los valores se encuentran en la misma tabla del Nodo.
iXML: para alcanzar el valor de una instancia es necesario en este modelo consultar previamente la tabla Estructura.

XMI vs ToXin

- ToXin utiliza estructuras de almacenamiento basadas íntegramente en memoria, por lo tanto se dispone de una mayor flexibilidad en la implementación. Además, el procesamiento sobre memoria es sumamente veloz comparado con procesos que acceden a disco. Por otro lado, la característica volátil de la memoria de una computadora hace que todo el trabajo de indexación se pierda en caso de falla en el equipo. Finalmente como desventaja de la implementación sobre memoria podemos mencionar que el rendimiento del índice se encuentra totalmente determinado por la cantidad de memoria física del equipo. En caso de que el tamaño de la misma sea excedido, el proceso de paginación de memoria a disco dañaría seriamente la performance del proceso de acceso a datos vía ToxIn.
  XMI fue especialmente diseñado para funcionar sobre una base de datos del tipo RDBMS. Esto posibilita el almacenamiento de un volumen mucho mayor de documentos. Además los índices se encuentran almacenados en disco, por lo que no son afectados por fallas en la computadora. En el caso de actualizaciones al índice, la implementación sobre una RDBMS contempla la administración de accesos concurrentes y el aseguramiento de la consistencia en los datos, cosa no contemplada en la implementación de ToxIn.

- ToXin mantiene una tabla (en realidad un B+ Tree) por cada uno de los paths o elementos del documento. Esto es complicado de administrar en una base de datos relacional por la necesidad de crear tablas en el momento de cargar un documento y de hacer crecer de manera desproporcionada la cantidad de tablas de la base de datos. Por el contrario XMI utiliza una sola tabla relacional para almacenar todos los paths de todos los documentos indexados, permitiendo crear índices sobre esta tabla para acelerar la búsqueda.

- Para consultas del tipo //Libro, siendo libro un tag del documento y la consulta devolviendo todos los tags Libro en cualquier path que se encuentren, ToxIn debe

recorrer todas las tablas de valores e instancias a fin de localizar si el tag libro se encuentra presente en el path de dicha tabla.
En el caso de XMI, para obtener el resultado de esta consulta, solo es necesario acceder a la tabla de estructura (indexada por el campo tag) y de allí obtener todas las ocurrencias de este tag en las tablas de datos.

- ToxIn almacena todo valor incluido en el documento como string. Esto desperdicia espacio de almacenamiento y hace menos eficiente la manipulación de cada tipo de datos específico. Por otro lado se hace muy difícil establecer predicados sobre los valores de los tags, si los mismos son solo cadenas de texto. Por ejemplo, como consultar por cosas tales como Precio>500.
  Si consideramos el modelo XMI, una de las bases de su diseño es separar los datos de acuerdo a su tipo. Esto elimina los problemas mencionados para ToxIn, haciendo más eficiente la tarea de indexación y acceso a datos.

- Espacio de disco utilizado para almacenar los índices se mantiene lineal al tamaño del archivo. EL minimizar el espacio de almacenamiento fue uno de los objetivos del diseño de Toxin. También lo fue para XMI, optimizando aún más el almacenamiento a través del uso de tablas individuales para cada tipo de datos soportado.

- ToXin no considera los links de los IDRefs.

- En ToXin no existe forma rápida de acceder a un nodo dada su clave / instancia, para ello es necesario recorrer todas las tablas de valores e instancias en busca del nodo. En XMI basta con acceder a la tabla de instancias, tomar su clave de esquema, y con esta información acceder a la tabla de estructuras para conocer en que tabla de datos se encuentra y el valor del nodo y obtenerlo desde allí. Esta diferencia se hace particularmente marcada en el caso de documentos con mucha cantidad de tags o paths distintos

- Ambas implementaciones utilizan el modelo DOM y un parser de terceros para reconocer el documento XML ToxIn utiliza el parser de IBM XML4J y XMI utiliza el MSXML parser de la empresa Microsoft.

- No queda claro si ToxIn permite indexar múltiples documentos, algo que XMI maneja sin problemas.

ACCESO RAPIDO A DATOS DE UNA INSTANCIA

En ToXin no existe forma rápida de acceder a un nodo dada su clave / instancia, para ello es necesario recorrer todas las tablas de valores e instancias en busca del nodo. En XMI basta con acceder a la tabla de instancias, tomar su clave de esquema, y con esta información acceder a la tabla de estructuras para conocer en que tabla de datos se encuentra y el valor del nodo y obtenerlo desde allí. Esta diferencia se hace particularmente marcada en el caso de documentos con mucha cantidad de tags o paths distintos. ***Ya vimos que hay casos en que esto no parece tan bueno, sobre todo si hay un tipo de datos que prevalece.***

La cantidad de accesos necesarios para llegar a una instancia en el modelo ToXin es mayor al necesario en el modelo XMI.

Demostración:

Sea D_1 un documento XML bien formado con las siguientes características:

- T Cantidad de tags del documento
- P Cantidad de paths distintos dentro del documento

Sea X_1 la cantidad de accesos necesarios para alcanzar una instancia determinada (dada por una clave única) en una tabla de instancias o tabla de valores para ToxIn.

Sea X_2 la cantidad de accesos necesarios para alcanzar una instancia determinada (dada por una clave única) en la tabla de instancias de XMI para D_1.

Sea V_t la cantidad de tablas de instancias presentes en una representación del modelo ToxIn para D_1.

Sea I_t la cantidad de tablas de datos presentes en una representación del modelo ToxIn para D_1

$VI_t = V_t + I_t$ la cantidad total de tablas presentes en la representación de ToxIn

Sea T_i el diccionario de las tablas pertenecientes al documento D_1

Acceso a la instancia I usando ToXin

- Peor caso, cuando la instancia buscada se encuentra en la última tabla de instancias o valores consultada

  $A_{t1} = VI_t + (VI_t * X_1)$

- Caso promedio, cuando la instancia buscada en la tabla de instancias o valores cuyo ubicación corresponde a la media de la T_i

  $A_{t2} = (VI_t / 2) + ((VI_t / 2) * X_1)$

Acceso a la instancia I usando XMI

- Peor caso y caso promedio, el valor de la instancia se encuentra en la única tabla de instancias del modelo.

  $A_x = X_2$

Se puede afirmar que $X_1 \geq 1$ ya que será necesario al menos un acceso a través de la estructura de datos para llegar a la instancia buscada. Por otro lado, es posible acotar el valor de X_2 para documentos de T tags mediante cálculos relacionados a la estructura de un índice en una base relacional y la longitud de los datos almacenados en la tabla de instancias de XMI.

Por otro lado, es posible verificar que $X_2 \leq 2$ para documentos con una cantidad de tags cercana a los 100000 (y aún más grandes)

Ejemplo (documento sin paths repetidos)

Sea D_1 un documento XML tal que:

$T_{D1} = 321$
$P_{D1} = 321$

Sean

$V_t = 310$
$I_t = 11$
$VI_t = 321$

Para acceder a la instancia I serán necesarios en ToXin:

$A_{t1} = 321 + (321 * 1) = 642$ (Peor caso)

$A_{t2} = (321/2) + ((321 / 2)*1) = 321$ (Caso promedio)

Para acceder a la instancia I serán necesarios en XMI:

$A_{t1} = X_2 \leq 2$ (Peor caso y caso promedio)

Ejemplo (documento con pocos paths diferentes)

Sea D_1 un documento XML tal que:

$T_{D1} = 125000$
$P_{D1} = 5$

Sean

$V_t = 4$
$I_t = 1$
$VI_t = 5$

Para acceder a la instancia I serán necesarios en ToXin:

$A_{t1} = 5 + (5 * 1) = 10$ (Peor caso)

$A_{t2} = (5/2) + ((5 / 2)*1) = 5$ (Caso promedio)

Para acceder a la instancia I serán necesarios en XMI:

$A_{t1} = X_2 \leq 2$ (Peor caso y caso promedio)

Notas:

Cabe aclarar que en el trabajo denominado “The XML Web: a First Study” se dice que el promedio de profundidad de los documentos XML encontrados en la Web es de 4 niveles. Por lo tanto, la cantidad de paths distintos implementados como tablas en el modelo ToXin será al menos 4, pudiendo ser aún mucho mayor en el caso de existir distintos paths para cada nivel.

Capítulo II - Experimentos

Características generales de los documentos XML existentes en la actualidad

Detallamos a continuación algunas de las conclusiones más importantes presentadas en el trabajo denominado "*The XML Web: a First Study*" (*), cuya autoría pertenece a Laurent Mignet, Denilson Barbosa y Piernagelo Veltri. En el mismo se hace un estudio pormenorizado de las características de los documentos XML presentes en la WWW. Este estudio nos permite enunciar propiedades importantes para grandes cantidades de documentos, entre los cuales se encuentran aquellos que podrían ser implementados dentro de nuestro modelo.

Tamaño de los documentos: Los documentos XML que se encuentran en la red son pequeños, el tamaño promedio de los documentos es de 4Kb.

Anidamiento: El 99% de los documentos tiene menos de 8 niveles de elementos anidados. La profundidad promedio de los documentos es de 4 niveles.

Recursión: El 15% de los documentos XML en la Web tienen recursión. Mientras que la recursión es tomada naturalmente en los documentos XML (no generan volumen en los documentos XML ni en las especificaciones del esquema) tienen un impacto considerable sobre la performance de procesadores de consulta (query processors y mecanismos de almacenamiento para XML). El máximo número de niveles utilizado para la recursión de elementos en un documento simple es 9.

Estructura del documento: El volumen de tags y de atributos es alto con respecto al tamaño de los documentos. El número de atributos excede ampliamente el número de nodos. Se observó

en el trabajo que la información de estructura de los documentos XML es superior al contenido textual.

Las conclusiones antes mencionadas nos orientan en los posibles tipos de documentos a generar para la etapa de prueba.

Generación de archivos de prueba

Para la generación de archivos de prueba que serán utilizados al momento de evaluar la performance general del sistema hemos decidido utilizar la herramienta ToxGene de la Universidad de Toronto.

La misma nos permite la generación de archivos de distintas características, como documentos cuasi-relacionales, documentos con datos altamente desestructurados, etc. Cada uno de estos tipos de documento será ampliamente estudiado con respecto al comportamiento que la implementación de nuestro modelo presenta en su procesamiento.

Rendimiento del proceso de parsing

En esta sección se analiza el rendimiento presentado por el módulo de parsing de nuestra implementación del modelo XMI. El mismo está considerado para la carga de documentos de distinta clase: relacionales, estructurados, desestructurados, grandes, pequeños, con muchos o pocos tags distintos, etc.

Consultas

Proceso de consultas y actualización

Las consultas que se implementen permitirán realizar las siguientes operaciones

Consultas

- Reconstrucción de los objetos del documento XML
- Selección de valores con determinadas condiciones
- Predicado Opcionales
- Predicados sobre los nombres de los atributos

Observación:

La estructura del documento original se asumirá que cumple con la definición de un documento XML bien generado. En el caso que alguna actualización que se produzca sobre la base no se corresponda con un documento XML bien definido no se chequeará, permitiendo la actualización de la base de datos.

Tamaño de documentos XML a almacenar *(depende de las claves de cada una de las tablas de datos de almacenamiento)*

Interfaz de Consultas

A continuación se presentan las consultas posibles y válidas a realizar sobre los datos almacenados en XMI.

Los ejemplos se desarrollaron sobre la consulta realizada en el sitio "The Species Analyst" sobre datos de biodiversidad.

Objeto Documento

Métodos

- ObtenerClaves

  Parámetros: Path (string), Condición (string)
- Salida: Claves (Documento XML con las claves obtenidas en forma de tags)

Este método devuelve claves a partir de un cierto path dado, y que cumplen con la condición especificada. El documento de claves tiene formato de documento XML, donde las claves están representadas como tags.

Ejemplo:

ObtenerClaves ("response/records/record", ObtenerValor(Clave_2_1,Valor) = 'BROWN, XANTUSIA VIGILIS',2)

Salida: archivo Resultado.XML

Donde Resultado es:

-<Clave_resultado>

<Clave_2_1/>

<Clave_2_7/>

<Clave_2_8/>

<Clave_3_1/>

<Clave_3_2/>

<Clave_3_3/>

-<Clave_3_10>

<Clave_3_50/>

<Clave_3_55/>

<Clave_3_56/>

-</Clave_3_10>

-<Clave_3_15>

<Clave_3_60/>

<Clave_3_61/>

<Clave_3_62/>

<Clave_3_65/>

-</Clave_3_15>

-<Clave_3_70>

<Clave_3_71/>

<Clave_3_72/>

<Clave_3_74/>

-</Clave_3_70>

</Clave_resultado>

- ObtenerValorClave

  Parámetros: Clave (string)
- Salida: Datos (Documento XML con la estructura y valores originales)

Este método devuelve los valores de los nodos a partir de un cierto path dado. El documento de salida tiene formato de documento XML con la estructura y valores originales.

Ejemplo:

ObtenerValorClave ("response/records/record/3_10")

Salida: archivo Resultado.XML

Donde el archivo resultado es:

```
-<resultado>
  -<3_10>
      <_3_50>Animalia</_3_50>
      <_3_55>XANTUSIA</_3_55>
      <_3_56>VIGILIS</_3_56>
  </_3_10>

  -<_3_10>
      <_3_50>Animalia</_3_50>
      <_3_55>XANTUSIA</_3_55>
      <_3_56>VIGILIS</_3_56>
  </_3_10>

  -<_3_10>
      <_3_50>Animalia</_3_50>
      <_3_55>XANTUSIA</_3_55>
      <_3_56>VIGILIS</_3_56>
  </_3_10>
  -----

  -<_3_10>
      <_3_50>Animalia</_3_50>
```

<_3_55>XANTUSIA</_3_55>

<_3_56>ARIZONAE</_3_56>

</_3_10>

-<3_10>

<_3_50>Animalia</_3_50>

<_3_55>XANTUSIA</_3_55>

<_3_56>ARIZONAE</_3_56>

</_3_10>

<resultado>

- ObtenerValorDatos

  Parámetros: Path (string), Condición (string)
- Salida: Datos (Documento XML con la estructura y valores originales)

Este método devuelve los valores que cumplen con la condición especificada a partir de un cierto path. El documento de valores tiene formato de documento XML con la estructura y valores originales.

Ejemplo:

ObtenerValorDatos ("response/records/record/3_10","ObtenerValor(Clave_2_1, valor) ='B. MUSGROVE, XANTUSIA VIGILIS',0)

Salida: archivo Resultado.XML

Donde el archivo resultado es:

```
-<resultado>
    -<_3_10>
        <_3_50>Animalia</_3_50>
        <_3_55>XANTUSIA</_3_55>
        <_3_56>VIGILIS</_3_56>
    </_3_10>
</resultado>
```

- ObtenerEstructura

  Parámetros: Path (string), Alcance (entero)

  Salida: Estructura (Documento XML solo con los tags originales)

Este método devuelve la estructura del documento a partir del path dado. El documento de estructura tiene formato de documento XML, donde las claves están representadas como tags.

Ejemplo:

ObtenerEstructura ("response/records/record/3_15")

Salida: archivo Resultado.XML

Donde Resultado es:

```
-<resultado>
    -<3_15>
        <3_60/>
```

<3_61/>

<3_62/>

<3_65/>

-<3_70/>

<3_71/>

<3_72/>

<3_73/>

<3_74/>

<3_70>

<3_15>

</resultado>

Objeto Nodo

Propiedades

- Clave (string) (solo lectura)
- Valor (objeto) (lectura / escritura)

Métodos

- ObtenerClave

  Parámetros: Nodo (nodo)

  Salida: Clave (string)

Este método devuelve la clave del nodo.

Ejemplo:

  c = Clave

  n = Nodo

  c = ObtenerClave(n)

  con n = (Clave_3_50, 'Animalia')

  y como resultado c = 'Clave_3_50'

- ObtenerValor

  Parámetros: Nodo (nodo)

Salida: valor (objeto)

Este método devuelve el valor del nodo especificado.

Ejemplo:

v = Valor

n = Nodo

v = ObtenerValor(n)

con n = (Clave_3_50, 'Animalia')

y como resultado v = 'Animalia'

Ejemplo de las consultas sobre el documento Biblioteca.xml

Documento Biblioteca.xml

```
<Biblioteca>
        <Libro>
                <Titulo>Ficciones</Titulo>
                <Autor>Borges</Autor>
        </Libro>
        <Libro>
                <Titulo>La invención de Morel</Titulo>
                <Autor>Bioy Casares</Autor>
        </Libro>
        <Libro>
                <Titulo>Anaconda</Titulo>
                <Autor>Horacio Quiroga</Autor>
        </Libro>
</Biblioteca>
```

Consultas:

- ObtenerClaves ("Biblioteca/Libro", ObtenerValor(Título,Valor) = 'Ficciones')

Salida: archivo Resultado.XML

Donde Resultado es:

```
-<Clave_resultado>
<Clave_Título/>
        <Clave_Autor/>
</Clave_resultado>
```

- ObtenerValorClave ("Biblioteca")

Salida: archivo Resultado.XML

Donde el archivo resultado es:

```
<resultado>
        <Libro>
        <Titulo>Ficciones</Titulo>
        <Autor>Borges</Autor>
</Libro>
<Libro>
        <Titulo>La invención de Morel</Titulo>
        <Autor>Bioy Casares</Autor>
</Libro>
<Libro>
        <Titulo>Anaconda</Titulo>
        <Autor>Horacio Quiroga</Autor>
</Libro>
```

```
</resultado>
```

- ObtenerValorDatos ("Biblioteca/Libro/Título/","ObtenerValor(Autor, valor) =' Borges')

Salida: archivo Resultado.XML

Donde el archivo resultado es:

```
<resultado>
    <Titulo>Ficciones</Titulo>
</resultado>
```

- ObtenerEstructura ("Biblioteca")

Salida: archivo Resultado.XML

Donde Resultado es:

```
<resultado>
    <Biblioteca>
        <Libro>
            <Titulo/>
        <Autor/>
        </Libro>
    </Biblioteca>
</resultado>
```

Especificación de lenguaje de consultas XPath Simplificado

Las consultas se realizarán a partir de una interfaz WEB que acceda a los datos ya almacenados en el modelo XMI.
Las consultas válidas que se podrán realizar sobre los documentos XMI indexados, almacenados en el modelo de datos XMI corresponde con la siguiente especificación.
Los resultados de las consultas serán convertidos desde los datos almacenados en el modelo XMI a formato XML.

A continuación se especifica el lenguaje válido de consultas sobre le modelo XMI (Xpath (2) Simplificado)

Path válido:
\documento\path [$cond_1$] [@ att $cond_2$] \

La especificación del path válido es la siguiente:

cond ← op n $valor_n$ | opt $valor_t$

op_n ← = | <> | <= | >= | > | <

op_t ← = | <> | <= | >= | > | < | like

$valor_t$ ← (a | b | ... | z | A | B | ... | Z | 0 | 1 | ... | 9 | " " | * | _ | -)*

$valor_n$ ← 0 | 1 | ... | 9 | 1E | 2E | ... | 9E | 0.D | E.D

E ← (0 | 1 | ... | 9) $^+$

D ← (0 | 1 | ... | 9) $^+$

att ← (a | b | ... | z | A | B | ... | Z)+ (a | b | ... | z | A | B | ... | Z) *

path ← tag_f | tag\path\

tag ← (a | b |...| z | A | B |...| Z | 0 | 1 |...| 9)+ (a | b | ... | z | A | B | ... | Z | 0 | 1 | ... | 9 | " " | * | _ | -) *

tagf ← (a | b |...| z | A | B |...| Z | 0 | 1 |...| 9)+ (a | b | ... | z | A | B | ... | Z | 0 | 1 | ... | 9 | " " | * | _ | -) *

documento ← (a | b |...| z | A | B |...| Z | 0 | 1 |...| 9)+ (a | b | ... | z | A | B | ... | Z | 0 | 1 | ... | 9 | " " | * | | _ | -) *

Consulta ← \documento\path\(cond | λ) (cond att | λ)\

Tipos de Consultas

Todos los modelos

Consulta por Path
Consulta por Path con condición sobre valor
Consulta de Instancias por Tag en un único Path
Consulta de Instancias por Tag en diferentes Paths
Reconstrucción de Documentos

Exclusivo XMI

Consulta por Path con condición sobre atributo
Consultas de Datos de una Instancia

Selectividad de la consulta

Bajo
Alta

	Consulta por Path	Consulta por Path con condición	Consulta Instancias por Tag en único Path	Consulta Instancias por Tag diferentes Paths	Consulta datos de una instancia
Selectividad Baja	$Q1_{SB}$	$Q2_{SB}$	$Q3_{SB}$	$Q4_{SB}$	Q5
Selectividad Alta	$Q1_{SA}$	$Q2_{SA}$	$Q3_{SA}$	$Q4_{SA}$	

Tipos de Documentos

Características significativas:

Tamaño

Tamaño del archivo
Cantidad de tags

Distribución de datos

Datos relacionales
Datos no relacionales

Anidación

Muchos Niveles de Profundidad
Pocos Niveles de Profundidad

Estructura

Muchos Paths
Pocos Paths

Documento	Tamaño			Distribución	Anidación	Estructura
	Tamaño (kb)	Cantidad nodos	Cantidad valores			

NOTAS

Para Toxin en consultas Q3 y Q4 existe una demora por el tiempo que requiere acumular en algún tipo de estructura las instancias provenientes de distintas tablas de datos que poseen el tag buscado.

CAPITULO III – Conclusiones y trabajos futuros

Actualizaciones

- Insertas nuevos Objetos
- Insertar nuevos atributos
- Eliminar Objetos
- Eliminar atributos

ANEXO 0

ToxGene

Es necesario contar con una colección acotada de documentos XML que abarque documentos con diferentes complejidades y tamaños, por ejemplo documentos relacionales, homogéneos y grandes con muchas referencias entre sus elementos; documentos planos, como textos largos, para obtener los mejores y más variados resultados de nuestro trabajo.

Existen aplicaciones que nos permiten generar estos documentos en forma automática, algunos de ellos, como Toxgene[iii] donde se generan los datos y las estructuras de los documentos de manera aleatoria, lo que nos permite tener el control sobre la estructura de los documentos a utilizar. Otro generador de documentos XML es IBM XML Generator[iv], este generador necesita que se especifique en un Document Type Definition (DTD) la estructura y las características de los datos.

Decidimos realizar la generación del conjunto de documentos a utilizar en el proceso de prueba a través de ToXgene[v], un template extensible para la generación de XML desarrollado en la Universidad de Toronto como parte del proyecto ToX (Toronto XML Engine). ToXgene nos permite generar los datos a partir de un template en el que se describen los datos deseados. Tenemos el control total sobre los datos que se generen. Podemos generar documentos con estructuras irregulares, con elementos complejos, atributos y diferentes valores en los datos, además soporta diferentes distribuciones probabilísticas.

ToXgene soporta compartimento de datos, por ejemplo los valores de algunos elementos (o atributos) puede ser compartidos entre diferentes elementos (o atributos) en el mismo (o diferente) documento XML. Esto nos permite generar colecciones de documentos

correlacionados. Además puede producir datos conforme a las restricciones de integridad especificada por el usuario.

Para experimentar nuestro proyecto definiremos un conjunto de consultas que realizaremos sobre diferentes tipos de documentos XML. Utilizaremos el tiempo de respuesta de estas consultas como métrica de performance. Utilizaremos documentos con datos relacionales, documentos planos, complejidad de estructuras (grado de anidamiento) y tamaño de los valores.

Para cada uno de los distintos archivos de prueba generados, se presentarán sus características, como cantidad de tags, cantidad de caminos distintos, tamaño del archivo, distribución de datos, etc. Cada uno de estos archivos será posteriormente accedido por el motor de consulta a fin de evaluar la performance del mismo utilizando diversos tipos de documentos y variadas consultas.

ANEXO 1

Detalles de implementación

Para la implementación del modelo XMI se utilizan los siguientes elementos de software y hardware:

- **Servicio de datos**

  Motor de base de datos relacional (RDBMS) Microsoft SQL Server 2000 Enterprise Edition

- **Lenguajes de programación**

  - Microsoft Visual Basic 6.0 (motor de consulta)
  - Microsoft Visual Basic .NET 2003 (proceso de parsing)
  - Microsoft ASP .NET v1.1 (interface Web)

- **Servicios Web**

  - Microsoft Internet Information Server 6.0

- **Plataforma**

  - Microsoft Windows 2003 Enterprise Edition

- **Hardware**

    - Intel Pentium 4 1.5Ghz, memoria RAM 256Mb, disco rígido IDE (Ultra DMA 100) 7200 RPM.

ANEXO 2

Acceso a la implementación de XMI

Interfase Web

La implementación del modelo XMI se encuentra disponible en:

http://horacioag.dyndns.org/tesis

Por tratarse de un servidor no comercial, el mismo no se encuentra disponible en todo momento. Para acceder al sitio en funcionamiento, deberá solicitar una presentación vía e-mail con un día laborable de anticipación a la siguiente dirección de mail:

horacioag@speedy.com.ar

A través de la interfase web suministrada, es posible tanto el parsing y almacenamiento de un documento XML en nuestro modelo XMI sino también la consulta de este y otros documentos almacenados mediante el uso de las primitivas básicas (solo algunas se hayan disponibles)

[i]

[ii]

[iii] *ToXgene - the ToX XML data generator -. http://www.cs.torornto.edu/tox/toxgene - 2001.*

[iv] *IBM XML Generator - http://www.alphaworks.ibm.com/tech/xmlgenerator*

[v] *ToXgene - Barbosa, Mendelzon, Keenleyside y Lyons - ToxGene: An extensible template-based data generator for XML - 2001*