INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

Módulo XML para acceder al Sistema Administrador de Bases de Datos SQLmx a través de Internet

T E S I S

QUE PARA OBTENER EL GRADO DE MAESTRA EN CIENCIAS DE LA COMPUTACIÓN CON ESPECIALIDAD EN SISTEMAS DE INFORMACIÓN Y BASES DE DATOS

PRESENTA:

GLORIA IRENE TREJO SOTO

DIRECTORA: M. EN C. SANDRA DINORA ORANTES JIMÉNEZ

MÉXICO, D.F. **MAYO, 2002**

Dedicatorias

A La Gloria del Gran Arquitecto del Universo,
pues nada mejora lo que él ha hecho.

A mis Papás: Gloria y Cutberto:
Por darme todo y una vez más dejarme en
libertad para construir mi destino.

A mis hermanos: Luis Manuel y Cutberto
Por su constante apoyo y cariño.

A mis tíos: Margarita y Gerardo
Quienes han sido como mis padres.

A José Manuel ;)

Agradecimientos

Gracias Dios por permitirme llegar hasta aquí.

La realización de esta tesis es producto del apoyo de muchas personas, algunas con conocimientos, otras con apoyo moral, pero todos sin duda hemos tratado de dar lo mejor para lograr el objetivo de la misma.

Me gustaría agradecer al **Instituto Politécnico Nacional**, por darme la oportunidad y el orgullo de pertenecer a él; al **Centro de Investigación en Computación** por permitirme realizar la Maestría en Ciencias de la Computación con la especialidad de mi agrado: Sistemas de Información y Bases de Datos, y al laboratorio con el mismo nombre en el cual he trabajado.

Entre las personas a las que agradezco su apoyo se encuentran la *M. en C. Sandra Dinora Orantes Jiménez* quien amablemente y con dedicación ha dirigido esta tesis, al igual que el *M. en C. Alejandro Botello Castillo* quién me dio la oportunidad de trabajar con él como tesista y me asesoró hasta el término del trabajo que aquí se detalla, ambos me brindaron invaluablemente su tiempo, conocimientos, compañerismo y dirección en el desarrollo de este tema.

También quiero agradecer al Dr. Figueroa Nazuno, al Dr. Agustín Gutiérrez Tornés, al M. en C. Gilberto Martínez Luna y al M. en C. Alfonso May Arrioja, sus aportaciones que como sinodales hicieron, revisando este documento y su software de aplicación, y gracias a ellas pude complementar el trabajo que presento.

Me gustaría nombrar a personas a las que respeto y aprecio mucho: mis maestros; especialmente al M. en C. César Saúl Guzmán Rentería, quien me brindó su amistad y me dio la oportunidad de colaborar con él; al Dr. Adolfo Guzmán Arenas por darme apoyo y ánimo; Dr. Hugo César Coyote, Dr. Agustín Gutiérrez, M. en C. Rafael Cen Zubieta, M. en C. Ivonne Rivera, M. en C. Germán Téllez, M. en C. Tomás Maldonado, M en C. Gilberto L. Martínez, M. en C. Enrique Arista, M. en C. Alejandro Botello; pues con sus clases compartieron conmigo sus conocimientos.

Otras personas que tal vez sin saberlo me han hecho llegar hasta aquí, son mis amigos. Gracias a todos por su alegría, cariño, tiempo, amistad y uno que otro Oso.

También quiero agradecer el amor y cariño, de mi familia, que aunque distante físicamente, siempre estuvo conmigo dándome ánimo para seguir adelante; de José Manuel, por sacrificar su tiempo para que yo terminara la tesis.

Finalmente quiero agradecerte a tí, por el interés en el tema.

Contenido

Lista de Figuras

Capítulo 2

Capítulo 3

Capítulo 4

Capítulo 5

Lista de Tablas

Módulo XML para acceder al Sistema Administrador de Bases de Datos SQLmx a través de Internet

Resumen

La importancia de la conectividad de un Sistema Administrador de Bases de Datos, particularmente si se desea accederlo desde Internet, se debe a que los sistemas operativos y equipos con que cuentan los usuarios son diversos, para solucionar este inconveniente se emiten estándares y recomendaciones por parte de organismos facultados, con el fin de lograr una mejor conectividad y acceso a la información.

XML (***eXtensible Markup Language***, Lenguaje de Marcas Extensible), es la recomendación propuesta por el W3C (*World Wide Web Consortium*, Consorcio de la WWW) como un metalenguaje de marcas para estandarizar el formato de publicación en Internet; su característica de *independencia del modelo de datos* permite manipular información procedente de bases de datos y visualizarla desde un navegador Web.

SQLmx es un Sistema Administrador de Bases de Datos Relacional, desarrollado en el Centro de Investigación en Computación, y el Módulo XML que se detalla en esta tesis: se agrega como un componente de consulta y modificación hacia los datos que almacena, extendiendo sus capacidades para recibir a través de Internet sentencias SQL (***Structured Query Language***, Lenguaje Estructurado de Consulta) de manipulación de datos mediante documentos con formato XML.

Abstract

XML Module to access SQLmx Database Management System through Internet

The importance of a Database Management System connectivity, particularly if we want to access it from Internet, it's due to the great diversity of the operating systems with count the equipments user. Consequently standards and recommendations are emitted by authorized organizations, in order to obtain a better connectivity and access to the information.

XML (eXtensible Markup Language), is the recommendation proposed by the W3C (World Wide Web Consortium) as a metalanguage of marks for the standardization in the format to the publication in Internet; this data model with independence characteristic permits to manipulate information originated in databases and visualizing it on any Web browser.

SQLmx is a Relational Database Management System developed at the CIC (Center of Computing Research, ***Centro de Investigación en Computación****). The XML module was developed in this thesis: details is added like a query and modification module of the database content, extending it to receive through Internet documents with XML format.*

Capítulo 1

Introducción

1.1 Antecedentes

En un principio, la Administración de Bases de Datos no estaba automatizada, pues se llevaba a cabo por medio de archiveros que almacenaban gran cantidad de carpetas con documentos. Este tipo de almacenamiento crece de una manera rápida, por lo que una búsqueda entre tantos papeles resultaba tediosa y cansada.

En algunas áreas, como en la biblioteconomía, se utilizaban archiveros con fichas bibliográficas. Una desventaja era el almacenamiento redundante, debido a que las fichas se diferenciaban únicamente por el orden de la presentación de los datos que contenían (por ejemplo: una clasificación por autor, otra por tema, etc.). Otra desventaja era el acceso a cada ficha: en ocasiones, cuando alguien deseaba consultarla, otra persona ya estaba utilizando el archivero que la contenía, por lo que había que esperar turno, dando como resultado grandes filas de espera.

La información se multiplica rápidamente y no se puede reducir, pues se requiere guardar datos históricos, por ejemplo, para la elaboración de estadísticas, estudios de mercado, minería de datos, entre otros. Como una solución a estos problemas, surge la automatización de la información y los sistemas de bases de datos, que se diseñan para administrar grandes cantidades de información.

La administración de los datos implica la definición de estructuras para almacenar la información (llámese modelo relacional, jerárquico, de red o de objetos), así como proporcionar mecanismos para la manipulación de la misma [1]. Las bases de datos permiten guardar información de manera rápida, eficiente, sin redundancias, consistente, en forma atómica y perdurable, lo cual brinda integridad, capacidad de recuperación, concurrencia y seguridad.

En la actualidad, las bases de datos son importantes en casi todas las áreas de aplicación en donde se usen computadoras, como en los negocios, la ingeniería, la medicina, las finanzas y la educación, entre otras. Existen bases de datos que almacenan lenguaje natural, datos multimedia, análisis de datos estadísticos y *metadatos* (datos sobre los datos); en ocasiones es tanta la información, que se hace

necesario implementar bases de datos distribuidas, replicadas y federadas para una mejor organización.

El desarrollo de un Sistema Administrador de Bases de Datos (SABD) permite analizar y diseñar herramientas con las que se puede trabajar libremente (modificar su estructura o agregarle funcionalidad), lo que ayuda a comprender su composición, su funcionamiento y la forma de implementarlo, de tal manera que se logre dar respuesta a las necesidades de manejo de información que se presenten.

El Centro de Investigación en Computación del Instituto Politécnico Nacional, a través del Laboratorio de Sistemas de Información y Bases de Datos, desarrolla un proyecto para la construcción de un SABD experimental: SQLmx, el cual se basa en el modelo relacional y hace uso del SQL (*Structured Query Language*, Lenguaje Estructurado de Consultas), para la administración de la información.

SQL incluye sentencias para la definición de datos, usando un DDL (*Data Definition Language*, Lenguaje de Definición de Datos), la modificación y consulta de datos, usando un DML (*Data Management Language*, Lenguaje de Manipulación de Datos) y la especificación de restricciones de datos. Los tipos de consultas, varían desde accesos a una sola tabla hasta aquellas que incluyen reuniones entre varias.

Hasta el momento, SQLmx soporta consultas mediante una API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones) escrita en lenguaje C, que funciona como un controlador ODBC (*Open Database Connectivity*, Conectividad Abierta de Bases de Datos), y mediante un controlador JDBC (*Java Database Connectivity*, Conectividad Java a Bases de Datos) para acceso a bases de datos usando el lenguaje Java, mostrando los resultados como una página HTML (*HyperText Markup Language*, Lenguaje de Marcas de Hipertexto) para visualizarlos desde un navegador de Internet.

La toma de decisiones en la mayoría de las empresas e instituciones modernas depende no sólo de las bases de datos, sino de la capacidad de compartir información útil a través de Internet, de tal forma que esté disponible independientemente de la plataforma y equipo utilizado para accederla, por lo que se han integrado organismos que regulan las propuestas de nuevas tecnologías de Internet, tal es el caso del W3C (*World Wide Web Consortium*, Consorcio de la WWW), que decidió normar los formatos de los documentos utilizados para publicar información en la Web, a través de un subconjunto del SGML (*Standard Generalized Markup Language*, Lenguaje de Marcas Generalizado Estándar), conocido como XML (*eXtensible Markup Language*, Lenguaje de Marcas Extensible).

Uniendo las ventajas de un SABD con las características de XML, se pueden recuperar documentos con etiquetas XML como resultado de consultas a la base de datos, preservando la información original del esquema de la base de datos, como son los nombres de las columnas, sus tipos de datos y la información acerca del resultado

obtenido; por ejemplo, la cantidad de registros que contiene una tabla, la tabla a la que pertenecen las columnas, entre otros. Esto le permite al cliente utilizar el resultado en la elaboración de reordenamientos, cambios en los tipos de letras, colores, etc. (además de visualizarlos en pantalla), sin tener que acceder nuevamente al servidor para cambiar la presentación de los datos.

Con base en lo anterior, surge el presente proyecto de tesis titulado **Módulo XML para acceder al Sistema Administrador de Bases de Datos SQLmx a través de Internet**, que da al SQLmx la capacidad de responder a las consultas de usuarios mediante el envío de documentos en formato XML, o al contrario, recibir documentos con datos en formato XML para actualizar la base de datos.

En el Marco Teórico, Capítulo 2, se explican los aspectos de la tecnología que se utiliza para la elaboración del sistema; el Capítulo 3, contiene el análisis y diseño del sistema, mientras que en el Capítulo 4 se describe su implementación; en el Capítulo 5, se muestran las pruebas y resultados, y finalmente el Capítulo 6 incluye las conclusiones de la tesis y trabajos futuros.

1.2 Planteamiento del Problema

La diversidad de visualizadores de páginas Web ha llevado a los organismos facultados a emitir recomendaciones y estándares, para tratar de solucionar incompatibilidades que se presentan en las marcas de texto, utilizadas por los productos comerciales.

Actualmente, la recomendación XML propuesta por el W3C busca que todo documento sea compatible con ella, estandarizando la forma de etiquetar documentos electrónicos para visualizarlos desde cualquier sistema operativo y equipo utilizado, y acceder a ellos a través de Internet sin alterar su presentación. El SABD SQLmx no cuenta con un módulo que le permita elaborar documentos XML como respuesta del acceso a la base de datos.

1.3 Objetivos

1.3.1 Objetivo General

- Diseñar un módulo usando el estándar XML para acceder al Sistema Administrador de Bases de Datos SQLmx a través de Internet.

1.3.2 Objetivos Específicos

- Implementar un módulo para el SQLmx que acepte consultas SQL y regrese los datos en un documento con formato XML.

- Implementar un módulo para el SQLmx, que acepte documentos basados en XML y haga actualizaciones (inserciones, eliminaciones y modificaciones) a la base de datos mediante sentencias SQL.

1.4 Justificación

Las bases de datos han evolucionado conforme a las necesidades de los usuarios. Mediante Internet, poco a poco cada individuo o institución va teniendo acceso a una mayor cantidad de información de las diversas ramas de la ciencia, con formatos distintos de almacenamiento y transmisión. En la actualidad, surge la posibilidad de utilizar aplicaciones que permiten acceder a información de forma dinámica, tal como a bases de datos con contenidos y formatos diversos.

Un SABD tiene como uno de sus propósitos realizar consultas eficientes, la forma en que se presentan los resultados de estas consultas también debe ser sencilla y fácil de entender, de modo que se obtenga un mejor aprovechamiento de los mismos: para graficarse, cuantificarse o procesarse posteriormente.

El Módulo XML, permite a un cliente realizar consultas al SQLmx haciendo uso del protocolo HTTP, retornándole los resultados mediante un documento XML; también es posible realizar actualizaciones que extraen los datos de un archivo XML para insertarlos, modificarlos o eliminarlos de una base de datos. Con este módulo, SQLmx podrá compartir datos por Internet y comunicarse con otras aplicaciones que utilicen el estándar XML, independientemente del sistema operativo o del lenguaje de programación utilizado.

1.5 Beneficios esperados

Con el desarrollo del Módulo XML se esperan los siguientes beneficios:

- Ampliar las capacidades de interoperabilidad del SABD SQLmx utilizando el estándar de intercambio de información en el Web, XML.

- Extender la funcionalidad de SQLmx para realizar manipulación de datos a través de Internet.

- Permitir al usuario presentar los datos resultantes de consultas a tablas con el estilo que considere pertinente.

- Contar con una arquitectura actual de bases de datos en el uso de tecnología de intercambio de datos a través de Internet.

1.6 Alcances y límites

Se tienen contemplados los alcances y limites siguientes:

Alcances:

- Utilizar sentencias SQL de manipulación de datos (véase Tabla 1.1).
- Recibir sentencias SQL para consultar la base de datos y generar documentos XML con los datos del resultado.
- Generar de forma dinámica DTD (*Document Type Definition*, Definición de Tipo de Documento) internas correspondientes a los documentos XML construidos como resultado de las consultas.
- Recibir documentos XML que contienen los datos a insertar, realizar el proceso correspondiente y hacer tantas inserciones como registros haya en el documento XML.
- Modificar o borrar registros, afectando a uno o más de ellos en la tabla si los datos de la condición coinciden con más de uno.
- Utilizar la API DOM (*Document Object Model*, Modelo de Objeto de Documento) para representación de los documentos XML.
- Desarrollo de una interfaz para acceder al SABD SQLmx a través de Internet.

Sentencia	Descripción
Lenguaje de Manipulación de datos (DML)	
SELECT	Recupera datos de la base de datos.
INSERT	Añade nuevas filas de datos a una tabla.
DELETE	Suprime filas de una tabla de la base de datos.
UPDATE	Modifica filas de una tabla.

Tabla 1.1 Sentencias SQL de manipulación de datos [2].

Límites:

- Esta aplicación se limita a las sentencias SQL de manipulación de datos (véase Tabla 1.1), no da soporte a sentencias de definición de datos, control de acceso, control de transacciones, ni SQL programático.

- La DTD se genera de manera interna; sin embargo, el programa cuenta con la funcionalidad de generarla externamente, haciendo la aclaración de que su almacenamiento ocupa espacio en el servidor y requiere un constante mantenimiento para su eliminación posterior.

1.7 Resumen

En este capítulo, se ha presentado la introducción del documento, que lleva por tema Módulo XML para acceder al Sistema Administrador de Bases de Datos SQLmx a través de Internet, situándose en la problemática que se quiere resolver y dando los objetivos y alcances esperados en esta tesis.

Capítulo 2

Marco Teórico

2.1 Introducción

Internet es una tecnología que se ha vuelto fundamental en el área de la Computación, ya que permite a un usuario acceder de manera transparente a información ubicada físicamente fuera de su alcance. El mundo de las bases de datos hace uso de Internet para llevar a cabo transacciones a distancia; actualmente la tecnología que permite publicar información estructurando mejor su contenido es XML. En este capítulo, se presentan los aspectos teóricos de las tecnologías SQLmx y XML utilizados en el desarrollo del Módulo XML.

2.2 Estado del Arte

2.2.1 Conectividad de bases de datos

Para que una aplicación pueda interactuar con una base de datos es necesario establecer una forma de comunicación. Para ello, se han desarrollado diversas maneras de conectarse a bases de datos buscando la eficiencia y sencillez, entre ellas ODBC (*Open Database Connectivity*, Conectividad Abierta a Bases de Datos), OLE DB (*Object Linking and Embedding Database*, Base de Datos con Vinculación e Incrustación de Objetos), y JDBC (*Java Database Connectivity*, Conectividad Java a Base de Datos).

ODBC.- Conectividad Abierta a Bases de Datos

ODBC, es una API (*Application Programming Interface*, Interfaz de Programación de Aplicaciones) de Microsoft para utilizar datos en SABD en varios ambientes operativos, como Windows y Linux, utilizando para ello SQL.

La arquitectura ODBC tiene cuatro componentes principales: la aplicación, un administrador de controladores, el controlador y la fuente de datos. El administrador de controladores es provisto por Microsoft junto con ODBC, y es una biblioteca de enlace dinámico que carga los controladores que implementan las llamadas a funciones ODBC e interactúan con la fuente de datos.

La interfaz se compone de los siguientes grupos de funciones:

Grupo de reservación y liberación de memoria. Este grupo define un manejador de ambiente, uno de conexión y uno de sentencia. Cada uno tiene las funciones de reservar el almacenamiento en memoria, para información de definiciones de conexiones a bases de datos, la memoria para una conexión en particular y la memoria para información de una sentencia SQL, respectivamente.

Grupo de conexión. Una vez que se ha reservado el ambiente, se pueden definir uno o más manejadores de conexión. Con esto se puede establecer una conexión al servidor, la cual se cierra cuando la aplicación deja de utilizarla.

Grupo de ejecución de sentencias SQL. Este grupo se encarga de obtener la consulta de usuario y transmitirla hacia el servidor, para ejecutar las sentencias.

Grupo de recuperación de resultados. Este conjunto de funciones recupera los datos del resultado y la información sobre los datos, como sus atributos, la cantidad de registros afectados por la sentencia SQL, etc.

Control de transacciones. Permite efectuar o no una transacción.

Control de errores. Regresa información acerca de los errores asociados con un controlador [3].

En la conectividad de base de datos, se busca crear una capa de abstracción para tener una base de datos independiente del código de la aplicación que la utiliza. ODBC proporciona esta abstracción para lenguajes como C, C++ y herramientas de desarrollo como Delphi, PowerBuilder y Visual Basic. Desafortunadamente, ODBC no facilita la independencia de plataforma como Java [4], para solucionar este inconveniente Microsoft creó OLE DB.

OLE DB.- Bases de Datos con Vinculación e Incrustación de Objetos

OLE DB es un estándar multiplataforma de acceso a todo tipos de datos, y está basado en la tecnología COM (*Component Object Model*, Modelo de Objetos de Componentes). Es el sucesor del ODBC e incluye un puente para dar soporte a los controladores existentes de ese tipo.

ODBC fue creado para acceder a bases de datos relacionales. OLE DB, está diseñado para fuentes de información relacional y no relacional incluyendo fuentes de datos ISAM (*Indexed Secuencial Access Method*, Método de Acceso Secuencial Indexado) y bases de datos jerárquicas, correo electrónico y almacenamiento en archivos: texto, gráficos y datos geográficos, etc., mediante una colección de interfaces COM que encapsulan varios de los servicios de los sistemas administradores de datos. Estas

interfaces permiten la creación de componentes de software que implementan dichos servicios [5].

JavaSoft resuelve el problema de independencia de plataforma mediante una API sencilla de acceso a base de datos desarrollada en java: JDBC, en la que se consideran muchos de los aspectos relevantes de ODBC.

JDBC.- Conectividad Java a Bases de Datos

JDBC es un conjunto de interfaces de comunicación entre aplicaciones Java y fuentes de datos (comparte la característica de independencia de plataforma del lenguaje) implementadas de manera diferente por cada vendedor. Conecta un programa de usuario con la base de datos de forma transparente, sin importar el software de administración de base de datos que se utilice para controlarlo.

JavaSoft define las siguientes categorías de controladores:

Tipo 1:

> Estos controladores son una tecnología de puente para acceder a la base de datos, y requieren para su funcionamiento que se instale software en los sistemas del lado del cliente.

Tipo 2:

> Son controladores API nativos; es decir, el controlador contiene código Java que invoca métodos nativos de C o C++ provistos por los proveedores de bases de datos para accederlas.

Tipo 3:

> El controlador JDBC en el cliente usa *sockets* para llamar una aplicación en el servidor y convertir las peticiones del cliente en una API específica del controlador deseado.

Tipo 4:

> Este tipo de controlador se comunica directamente con la base de datos usando *sockets* de Java, y es proporcionado por el fabricante.

JavaSoft trabajó en conjunto con Intersolv y crearon un puente ODBC, que relaciona llamadas JDBC a ODBC, permitiendo que las aplicaciones Java puedan acceder a cualquier SABD que soporte ODBC. Por lo tanto, JDBC resuelve el problema de portabilidad de las API propietarias de bases de datos [4]; sin embargo, realizar transacciones a distancia a través de Internet demanda estructurar la información de

modo que su contenido pueda reutilizarse en otras aplicaciones; por consiguiente, se necesita separar el contenido de la presentación de un documento electrónico. Los accesos a bases de datos por parte del cliente, han incluido al XML como otra opción de intercambio de información.

2.2.2 Interacción cliente-servidor

En la actualidad, la mayoría de las aplicaciones que se utilizan en entornos empresariales están construidas en torno a una arquitectura *cliente-servidor,* en la cual, una o varias computadoras son los servidores, que responden a un número grande de clientes conectados a través de la red. Los clientes suelen ser computadoras personales de propósito general orientadas al usuario final; en ocasiones, los servidores son intermediarios entre los clientes y otros servidores más especializados (por ejemplo, servidores de bases de datos). Con el auge de Internet, la arquitectura *cliente-servidor* ha adquirido una mayor relevancia, ya que es el principio básico de funcionamiento de la World Wide Web: un usuario que mediante un *visualizador (cliente*) solicita un servicio (páginas HTML, etc.) a una computadora que funciona como *servidor*.

En sus orígenes, los servidores Web se limitaban a enviar una página HTML cuando el usuario la solicitaba pulsando sobre un enlace. La interactividad era limitada, ya que el usuario podía pedir archivos, pero no enviar datos al servidor para obtener una respuesta personalizada; para complementar estos servicios se han desarrollado tecnologías útiles, entre ellas: los guiones CGI *(Common Gateway Interface*, Interfaz de Compuerta Común*)* y los *servlet.*

Para que los guiones CGI funcionen, es necesario compilarlos para el sistema operativo del servidor; los *servlet* proporcionan las mismas ventajas del lenguaje Java en cuanto a portabilidad, ya que son una clase Java, y por tanto tienen todas las características del lenguaje. Una ventaja de los *servlet* respecto a los guiones CGI es el rendimiento; un guión CGI es necesario cargarlo en memoria cada vez que se hace una petición de servicio por parte de los clientes; en cambio, un *servlet* queda activo en la memoria del servidor desde la primera vez que se llama, hasta que un programa que controla al servidor lo desactiva [31].

Otras tecnologías recientes de este tipo son las ASP (*Active Server Page*, Páginas Activas de Servidor) y las JSP (*Java Server Page*, Páginas Java de Servidor). Las ASP y JSP mezclan código con etiquetas de presentación de una página Web. El código se ejecuta del lado del servidor y presenta los resultados en la página dinámicamente. Generalmente, se utilizan cuando se desea conservar fija cierta parte de la página y cambiar algunos datos que se procesan cuando el cliente hace una petición.

El Módulo XML regresa resultados dinámicos que no conservan partes estáticas en el documento que se muestra como página al usuario; el medio de interacción *cliente-*

servidor utilizado en su desarrollo son los *servlet*, debido a que sus características satisfacen las necesidades de respuesta deseadas.

2.2.3 Aplicaciones comerciales afines

Los SABD comerciales ya incluyen utilerías como la que se propone implementar para el SQLmx. Productos como Oracle 8i de Oracle y SQL Server 2000 de Microsoft, cuentan con la capacidad de devolver resultados en XML o actualizar sus tablas extrayendo información de documentos con este formato.

Oracle XSU

El SABD Oracle incluye a partir de su versión 8i la utilería XSU (*XML SQL Utility*, Utilería XML SQL) que permite realizar lo siguiente:

- Transformar datos recuperados de tablas de bases de datos objeto-relacional o de vistas a XML.
- Extraer datos de documentos XML e insertarlos en las columnas respectivas de la tabla.
- Extraer datos de documentos XML, introducirlos, modificar o borrar valores de las columnas apropiadas.
- Soporta generación dinámica de DTD (*Data Type Definition*, Definición de Tipo de Documento).
- Permite la generación de documentos por medio de representación de objetos o de eventos (característica de Oracle 9i).

Como nuevas características de la versión 9i:

- Permite especificar que una columna en particular o un grupo de columnas sean representados como atributos XML en lugar de elementos.
- Proporciona soporte inicial de esquemas XML generados en línea para el resultado XML de cualquier consulta SQL.

La arquitectura que utiliza es la que se muestra en la Figura 2.1. El usuario puede introducir mediante una página Web un documento XML que es captado por el servidor Web, el cual lo pasa al servidor de aplicación y en él, la utilería XSU realiza el proceso correspondiente para enviar las sentencias SQL a la base de datos Oracle [6].

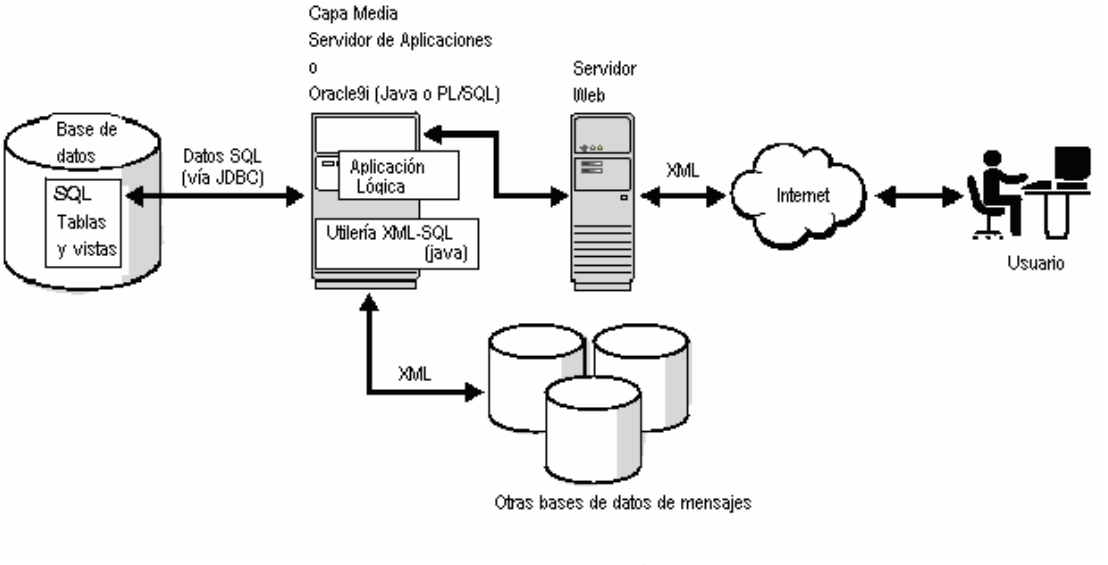

Figura 2.1 Arquitectura de la utilería XSU de Oracle.

SQL Server XMLSQL

SQL Server 2000 incluye una utilería llamada XMLSQL que permite dar soporte a XML:

- Accede a datos SQL Server 2000 en un formato XML.
- Accede a datos SQL Server 2000 utilizando HTTP.
- Utiliza la sentencia TSQL SELECT para realizar una selección FOR XML.
- Realiza consultas basadas en URL para acceder a los datos.
- Incluye un mecanismo para insertar, actualizar y borrar datos en una base de datos SQL Server 2000, utilizando información contenida en un documento XML [7].

La arquitectura que utiliza se muestra en la Figura 2.2, donde se tiene una aplicación cliente, que normalmente puede ser un documento HTML desplegado en el visualizador Internet Explorer (que contiene *guiones*), el cual, ejecuta una petición GET o POST de HTTP para acceder a la fuente SQLXML. Cuando la petición llega al servidor IIS (*Internet Information Server*, Servidor de Información de Internet), éste examina la dirección URL e identifica el directorio virtual que se desea consultar. Se invoca el filtro ISAPI (*Internet Server Application Programming Interface*, Interfaz de Programación de Aplicaciones de Servidor de Internet) SQLXML para procesar la petición. El filtro ISAPI analiza la petición y determina si

es una consulta natural o si requiere un documento con plantillas, y entonces busca su esquema. El filtro continúa y llama al proveedor SQL-OLEDB, que es el proveedor de los datos utilizado para establecer la conexión con la base de datos SQL Server 2000.

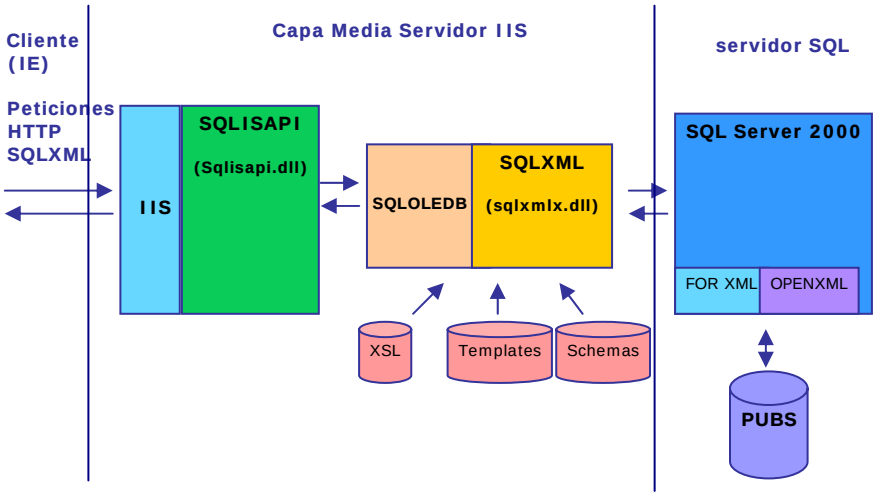

Figura 2.2 Arquitectura de la utilería XMLSQL de SQL Server 2000.

Cuando el comando pasa por SQL-OLEDB, este lo examina e identifica si es un comando específico para SQLXML, entonces invoca al componente SQLXML implementado como la biblioteca Sqlxmlx.dll. El componente SQLXML examina el comando y lo manda a las fuentes para ejecutar la operación. El componente SQLXML resuelve la operación y envía las sentencias correspondientes al SQL Server 2000, el cual recupera los datos de la base de datos solicitada.

Es necesario señalar que en México no se ha publicado investigación relacionada con este tipo de aplicaciones, por lo que la implementación del Módulo XML para el SABD SQLmx es una de las primeras aportaciones mexicanas de este tipo.

2.3 Características del SQLmx

SQLmx incluye los módulos básicos que componen a un SABD [8]: el sistema de archivos e índices, el diccionario de datos, el compilador de SQL (que incluye sentencias DDL y DML), el administrador de transacciones y los controladores ODBC y JDBC (véase Figura 2.3).

Entre las principales características se tienen:

- Permite el uso del un subconjunto del SQL/92, que incluye las sentencias para:

- La creación y eliminación de bases de datos, de tablas y de índices;
- La definición de los tipos de datos básicos como son enteros, numéricos, carácter, fechas y reales;
- La especificación de consultas multitabla;
- La especificación de consultas simples cuya condición involucre operaciones lógicas de comparación (=, < y >);
- Define reglas de verificación de integridad (usando la cláusula CHECK).

- Implementación y uso de un catálogo del sistema para la administración de *metadatos*.
- Capacidad de bloqueos a nivel de tablas y registros.
- Administración de los permisos de los usuarios dados de alta en el sistema.
- Búsqueda de información mediante el uso de rutas de acceso a los datos, como son las estructuras de árboles B+.
- Formulación de consultas a través de la Web mediante el uso de un controlador JDBC.
- Importación y exportación de datos al sistema en formato de texto. Entre algunas otras.

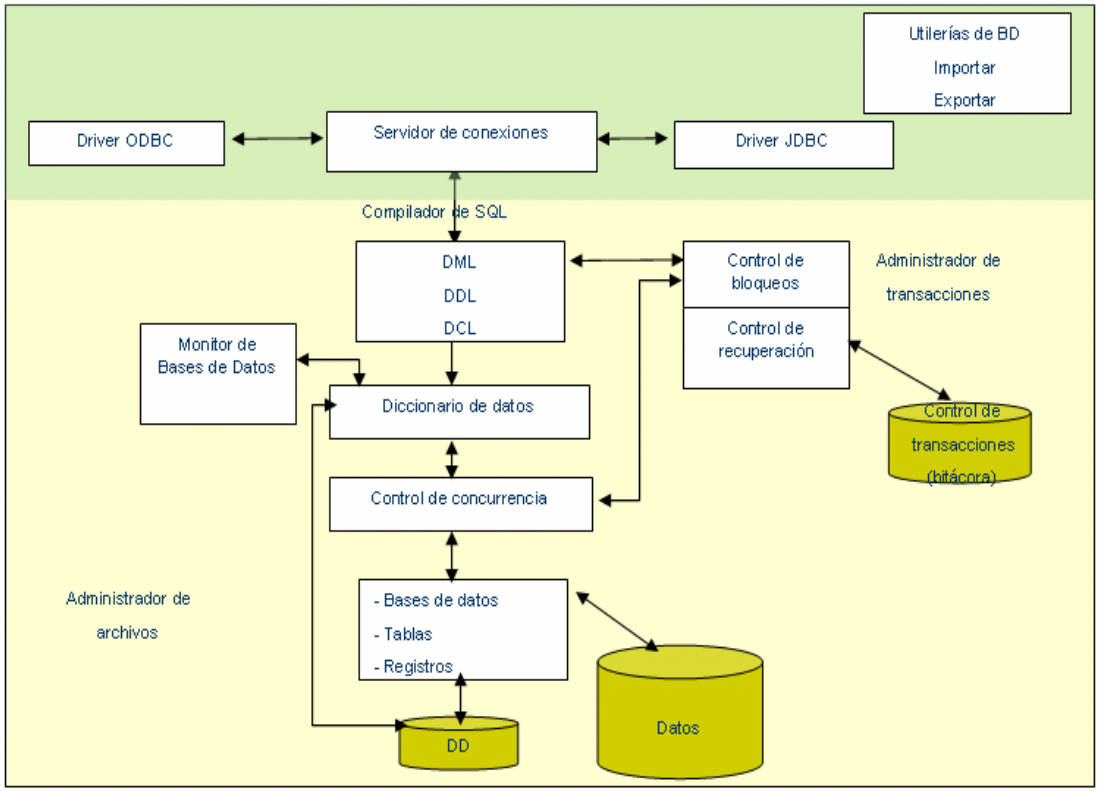

Figura 2.3 Arquitectura del SABD SQLmx.

El principal componente de todo sistema de información es su base de datos, por lo que la API que provee el SABD para su conectividad es muy importante, ya que de ella depende que un programa cliente pueda solicitarle servicios al SABD y obtener resultados.

2.3.1 El controlador JDBC del SQLmx

El controlador JDBC del SQLmx es de tipo 4 (véase sección 2.2.1 JDBC). Se comunica directamente con el SABD utilizando un protocolo propietario (véase Figura 2.4). Los controladores de protocolo nativo pueden ser incluidos en *applets* y cargados desde servidores HTTP. Sin embargo, el servidor de bases de datos puede ser instalado en la máquina en donde se ubica el servidor de Web [9].

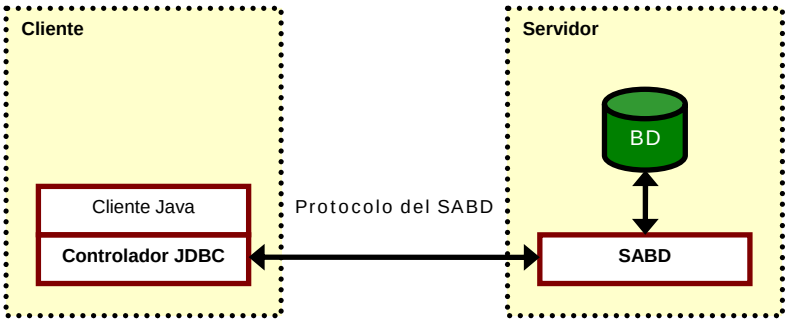

Figura 2.4 JDBC de protocolo nativo.

El Módulo XML hace uso de las clases de la API JDBC del SQLmx que se muestran en la Tabla 2.1.

Nombre de la clase	Descripción
Conecction	Permite conectarse a la base de datos.
Statement	Crea una sentencia SQL.
ResultSet	Obtiene los resultados de una consulta SQL.
ResultSetMetaData	Crea objetos que contienen los *metadatos* de un ResultSet.

Tabla 2.1 Clases del controlador JDBC del SQLmx a utilizar.

2.3.2 El Diccionario de Datos (DD)

Otro aspecto que se contempla para el desarrollo del Módulo XML es el Diccionario de Datos del SABD SQLmx, que almacena la información sobre las características de una base de datos, que se conoce como *metadatos*. Cuando se crea una base de datos, los datos que definen la estructura de las tablas que la componen se almacenan en

otras tablas (conocidas como el catálogo o tablas del sistema) que son administradas internamente por el sistema. El diagrama entidad-relación de la Figura 2.5 muestra el diseño del diccionario de datos del SQLmx. El Módulo XML consulta las tablas del sistema, para conocer los esquemas requeridos en la manipulación de datos al momento de realizar una consulta, inserción, modificación o eliminación en una base de datos (los tipos de datos del SQLmx se muestran en el Anexo A). Con los *metadatos* es posible generar la DTD útil para validar los documentos XML proporcionados por el usuario para actualizar una tabla, y el documento que se envía como resultado de una consulta al usuario cliente.

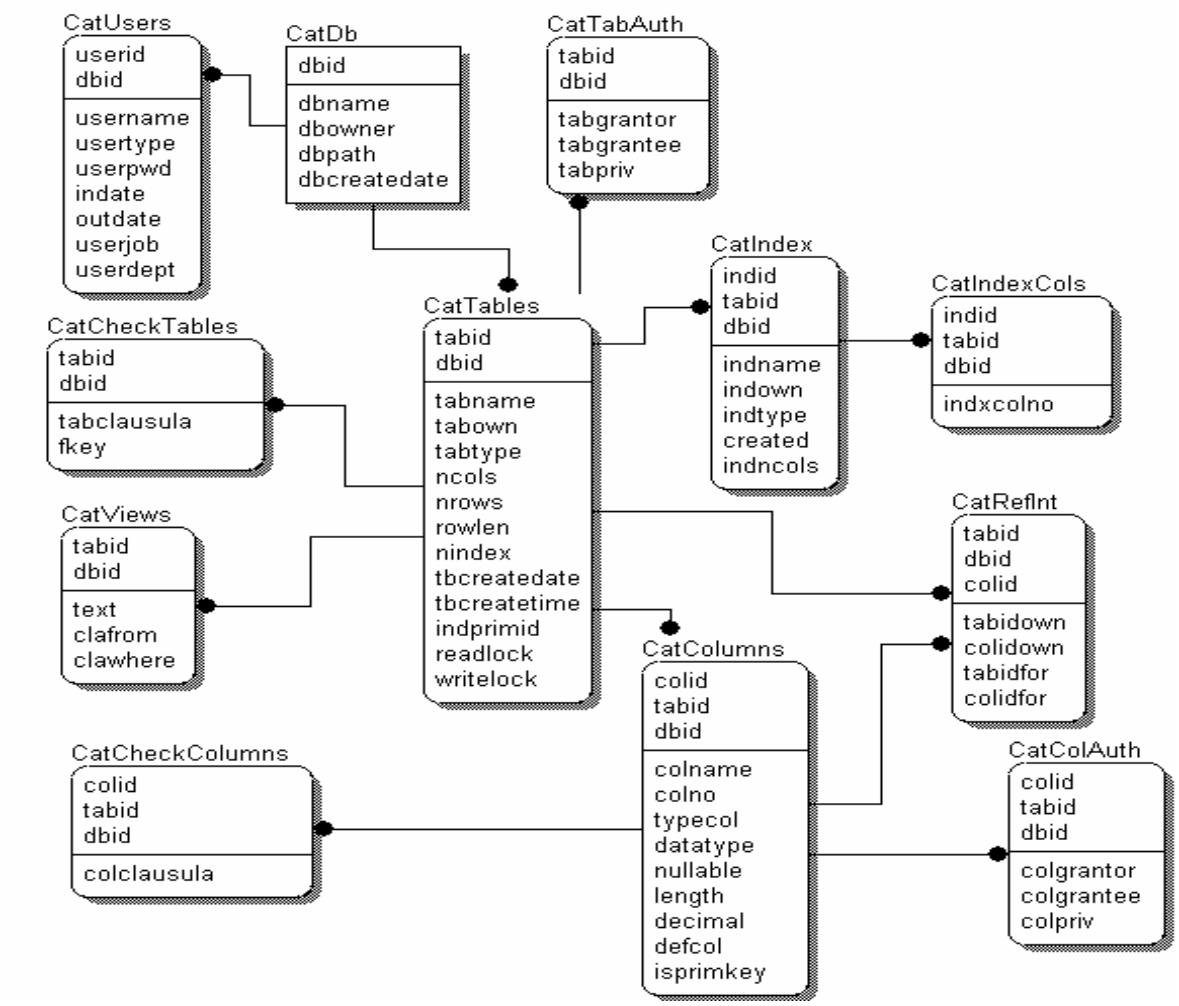

Figura 2.5 Diccionario de Datos SQLmx.

2.4 Características de la recomendación XML (*eXtensible Markup Language*, Lenguaje de Marcas Extensible)

2.4.1 Antecedentes históricos de XML

Los lenguajes de marcas son sistemas de descripción de información, normalmente documentos electrónicos, que son básicamente textos. Sin embargo, cada aplicación utiliza sus propios formatos, por lo que para abrir un documento en una aplicación distinta a la que lo creó, deben desarrollarse programas que conserven el diseño y presentación originales.

En los años 60, IBM resolvió los problemas asociados al tratamiento de documentos sobre diferentes plataformas, utilizando etiquetas con formato simple de texto, y lo denominó GML (*Generalized Markup Language*, Lenguaje Generalizado de Marcas), haciendo posible su tratamiento desde cualquier sistema que conozca el formato de las etiquetas.

ISO (*International Organization for Standardization*, Organización Internacional para la Estandarización) realizó una revisión del GML y lo convirtió en SGML (*Standard Generalized Markup Language*, Lenguaje de Marcas Generalizado Estándar), publicado oficialmente en la década de los 80 como ISO8879. SGML resuelve la incompatibilidad de etiquetas, pero el hecho de que sea complejo (400 hojas aproximadamente), ha dado como resultado el uso de sólo un subconjunto de las mismas, conocido como XML. Esta recomendación es un metalenguaje de marcas, es decir, un lenguaje para definir lenguajes de marcas, por lo que sus elementos describen lo que contienen, no la estructura física o presentación, como sucede con HTML.

HTML es el lenguaje de marcas más utilizado para la transmisión de datos en Internet. Entre sus características destacan:

- Posee una sintaxis simple con un conjunto fijo de etiquetas.
- Permite crear documentos multimedia al incorporar imágenes y sonido.
- Enlaza varios documentos entre sí con el uso de ligas [10].

HTML ha ido agregando etiquetas a su lenguaje desde su aparición, pero las páginas Web exigen mayor interoperabilidad, y debido a que el único organismo autorizado para añadirle etiquetas a este lenguaje es el W3C -si considera útil y de uso general la etiqueta propuesta como nueva- se recurre nuevamente a las partes útiles de SGML, el XML.

2.4.2 El Lenguaje de Marcas Extensible

XML es una recomendación del W3C desde febrero de 1998, basada en SGML; actualmente se encuentra liberada la versión 1.0, y establece un estándar que separa el contenido de un documento de su presentación, lo que hace posible visualizarlo en cualquier navegador Web sin depender del estándar de hojas de estilo o de un modelo de datos en particular. Otra ventaja que brinda es la independencia de plataforma: un documento XML se puede visualizar en una PC, en un dispositivo PDA o en una terminal de modo carácter. La utilidad de XML puede verse desde dos puntos diferentes:

- En aplicaciones que presentan información de documentos y son útiles para las personas en cuanto a qué representan, con etiquetas más descriptivas, su contenido, y
- En aplicaciones de datos que manejan información que va dirigida principalmente al software [11] (véase Figura 2.6).

Sin embargo, las reglas de XML son las mismas debido a que se concentra en el contenido del documento y esto lo hace independiente del medio de entrega, para el cual debe sufrir una transformación que lo adecue a los dispositivos existentes; por lo tanto, es posible editar y mantener documentos en XML y publicarlos automáticamente en diferentes medios electrónicos (véase Figura 2.7).

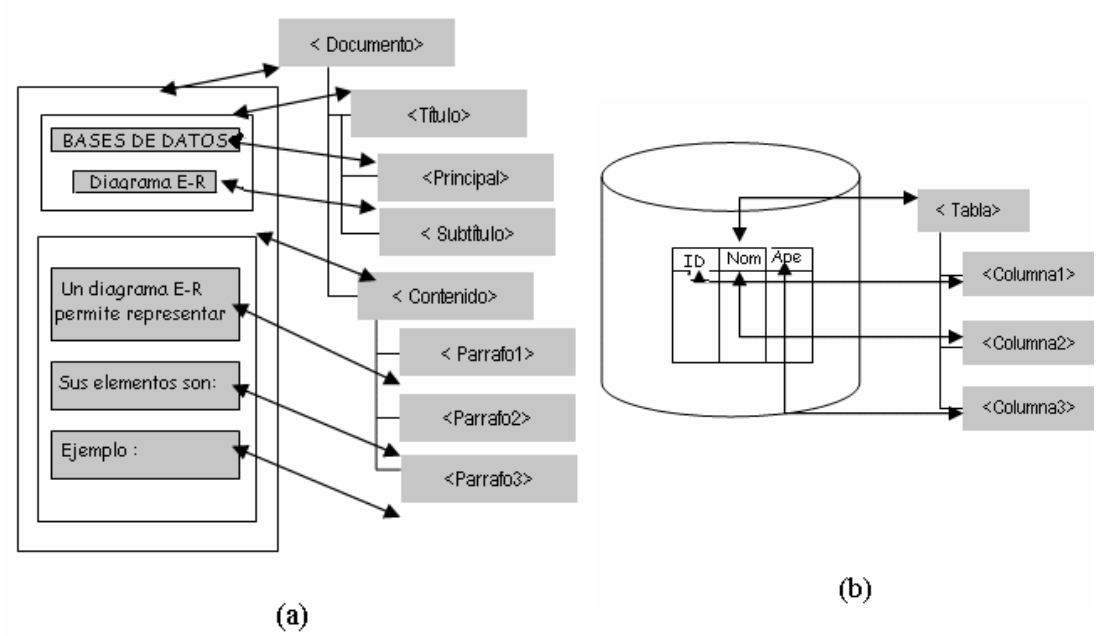

Figura 2.6 (a) Estructura de un documento XML dirigido a personas.
(b) Estructura de una base de datos en XML dirigida a personas y/o software.

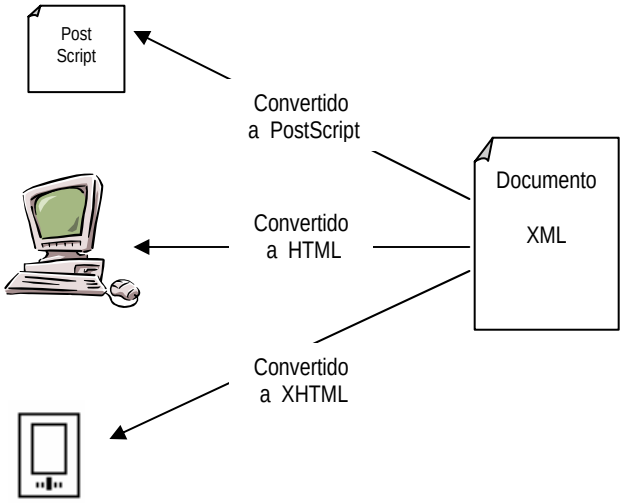

Figura 2.7. XML es independiente del dispositivo cliente.

XML está diseñado para ser escrito y editado como texto (con cualquier aplicación que soporte los caracteres *UNICODE*), lo que permite intercambiar datos entre diferentes aplicaciones sin perder información. Otra característica importante para utilizarlo es que provee documentos estructurados, de tal manera que sean comprensibles tanto por personas como por aplicaciones. Cada aplicación que utiliza XML tiene su propia sintaxis y vocabulario para elaborar documentos con ese formato, apegando la definición de sus etiquetas a las reglas de la recomendación XML.

Un documento escrito en XML no es un código fuente que se compile y se compruebe que tiene una sintaxis correcta; por lo tanto, tiene que ser procesado por otra aplicación que indique si está o no escrito correctamente (analizadores, editores, etc.).

Transmitir información con formato XML permite a un programa trabajar con los datos de un documento XML, procesando sus datos mediante herramientas y bibliotecas estándares, incluso utilizar diferentes herramientas para cada propósito: un tipo de programa para leer el documento, otro para editarlo, etc.

2.4.3 Estructura de un documento XML

XML se compone únicamente de texto, que representa marcas y contenido. Las marcas XML se encuentran encerradas entre paréntesis angulares (< >).

La primera declaración que aparece en un documento XML es opcional y sirve para que el procesador identifique el tipo de marcado del documento y la versión de XML

que se utilizó para crearlo, en algunos casos también incluye el tipo de codificación a utilizar y si el procesador debe llamar a otros documentos o si es independiente.

Esta declaración empieza siempre con los caracteres *<?xml* y termina con *?>*.

Ejemplo:

<?xml version=”1.0” encoding=”ISO-8859-1” standalone=”yes” ?>

2.4.4 Componentes básicos de un documento XML

Para efectos de este proyecto, solo es necesario utilizar los componentes básicos de XML (*elementos, atributos y comentarios*) [12], ya que con ellos se puede representar completamente la estructura de datos relacionales.

Elementos

Los elementos se utilizan para identificar las secciones de un documento XML y deben cumplir las siguientes reglas de nombrado:

- Los nombres se componen de uno o más caracteres sin espacios. Si un nombre consta de un único carácter, debe ser una letra mayúscula (A-Z) o minúscula (a-z).
- Un nombre sólo puede comenzar por una letra o por un guión de subrayado.
- A partir del primer carácter, se puede utilizar cualquiera de los caracteres definidos en el estándar UNICODE.
- Los nombres de elemento distinguen mayúsculas de minúsculas, por lo tanto, *NombreEmpleado, NOMBRE EMPLEADO,* y *nombreempleado* se consideran elementos diferentes.

Cada documento puede contener uno o más elementos, los cuales se delimitan por *etiquetas de comienzo* y *etiquetas de fin.* Cada elemento tiene un tipo, identificado por un nombre, y puede tener un conjunto de especificaciones de atributos, cada una de las cuales tiene un nombre y un valor, además que debe cumplir las restricciones siguientes:

- *Restricción de buena formación (RBF): Igualdad de tipo de elemento.* El nombre en la etiqueta de fin de un elemento, debe ser igual que el de la etiqueta de inicio.
- *Restricción de Validez (RV): Elemento Válido*. Un elemento es válido, si cumple con la regla que dice: “ningún elemento debe ser declarado más de una vez”.

Un elemento tiene la forma siguiente:

```
<NombreDelElemento>Contenido</NombreDelElemento>
```

Donde las *etiquetas de inicio* se delimitan por paréntesis angulares (como en <*NombreDelElemento*>), mientras que las *etiquetas de fin,* inician con los caracteres </ y terminan con el símbolo de > (como en </*NombreDelElemento*>).

Aunque las etiquetas de XML habitualmente delimitan un contenido, también pueden aparecer elementos sin ningún contenido, denominados *elementos vacíos (empty elements)*. En XML se puede representar un *elemento vacío* como sigue:

```
<NombreDelElemento/>
```

En XML los elementos pueden contener a su vez otros elementos (*elementos anidados*), pero sin solaparse.

Atributos

Un atributo describe un elemento mediante la asociación de pares `nombre-valor`. Las especificaciones de atributos pueden aparecer sólo dentro de las etiquetas de inicio y en las etiquetas de elemento vacío. Los valores de los atributos deben aparecer entre comillas dobles (").

Por ejemplo, para indicar que la altura de una persona se mide en metros se agrega el atributo *unidad.*

```
<AlturaPersona unidad="Mts">1.55</AlturaPersona>
```

Un atributo no puede declararse más de una vez en un elemento. Por lo tanto, el elemento siguiente no está bien formado, ya que no puede expresarse la altura al mismo tiempo en dos unidades diferentes:

```
<AlturaPersona unidad="Mts" unidad="cms">155</AlturaPersona>
```

Comentarios

Los comentarios pueden aparecer en cualquier punto del documento, fuera del resto de las marcas; además, pueden aparecer en la declaración de tipo de documento y en los lugares permitidos por la gramática. No forman parte de los elementos del documento. Un procesador XML puede, pero no debe, permitir a la aplicación obtener el texto de los comentarios. Por compatibilidad, la cadena "--" (dos guiones seguidos) no puede aparecer dentro de un comentario [12].

Los comentarios, son instrucciones incrustadas en un documento XML que proporcionan información adicional acerca del documento. Éstos utilizan la sintaxis que se muestra a continuación, y tienen ese formato para que las aplicaciones que procesen el documento puedan ignorarlos.

Ejemplo:

```
<!-- Esto es un ejemplo de comentario -->
```

2.4.5 Buena formación y validez de los documentos XML

Un documento *bien formado* es aquél que cumple con la Recomendación de la W3C, que estipula que un documento tomado como un todo tiene correspondencia entre sus etiquetas, cumple con todas las restricciones dadas en la especificación, y además, todas las entidades a las que hace referencia directa o indirectamente también están bien formadas [12]. Su *validez* se cumple si existe correspondencia con una DTD, que se debe incluir en la declaración <!DOCTYPE>.

Para poder procesar un documento XML es necesario un *parser*[1] que cubra la gramática del XML, revisando su buena formación y validez. Un *parser* no validador revisa la buena formación del documento XML, así como la de todos los documentos a los que se haga referencia en él -solo busca que se cumpla la sintaxis-; mientras que un *parser* validador, también se encarga de verificar que el documento XML cumpla con las especificaciones de la DTD y de los documentos a los que haga referencia, lo que garantiza el correcto significado semántico de las etiquetas. En el Anexo B se incluye una comparación de implementaciones de *parser*, de la cual se seleccionó Xerces de Apache para la realización de esta aplicación.

2.5 Interfaces para procesar un documento XML

Existen dos técnicas básicas para crear una interfaz entre el analizador y la aplicación: por medio de interfaces basadas en objetos e interfaces basadas en eventos.

2.5.1 Interfaz basada en objetos

Al utilizar una interfaz basada en objetos, el analizador crea una estructura de árbol de objetos que contiene todos los elementos existentes en el documento XML, permitiendo a la aplicación navegar a través de los nodos del árbol.

La Figura 2.9 muestra como se crea el árbol en memoria cuando el analizador lee el documento de la Figura 2.8.

[1] Aunque la palabra inglesa *parser* se traduce como analizador gramatical, en este documento se seguirá empleando como tal, ya que el autor considera que en el contexto del área de estudio es más claro su entendimiento.

```
<AlumList>
        <Alumno>
                <Nombre>Gloria</Nombre>
        <Alumno/>
        <Alumno>
                <Nombre>Alejandro</Nombre>
        <Alumno/>
        ..
<AlumList/>
```

Figura 2.8 Ejemplo de datos en XML.

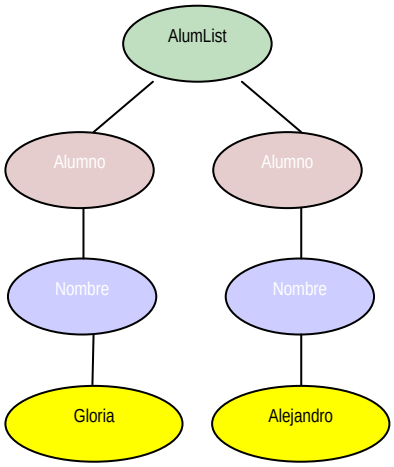

Figura 2.9 Representación basada en objetos (toma el ejemplo de datos del documento XML de la Figura 2.8).

La interfaz basada en objetos tiene la ventaja de poder navegar en el árbol, facilitando la consulta, inserción o eliminación de elementos en cualquier punto; sin embargo, esto presenta la desventaja de restringir el tamaño del árbol al espacio en memoria.

El W3C proporciona una recomendación estándar para construir una estructura de árbol en memoria para los documentos XML, llamada DOM (*Document Object Model,* Modelo de Objeto de Documentos) [13], para estructurar un documento. DOM facilita el acceso y la eliminación de componentes de manera independiente al lenguaje de programación (véase Anexo C).

Generalmente, DOM se añade como una capa entre el *parser* (véase Figura 2.10) y la aplicación que requiere la información del documento XML. DOM es utilizado por el nivel más alto de la aplicación, la cual puede disponer de la información e incluso ponerla en otro modelo de objetos propio, si lo desea [14].

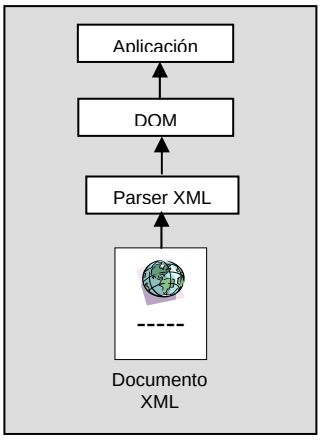

Figura 2.10 DOM (Modelo de Objeto de Documento).

2.5.2 Interfaz basada en eventos

Con una interfaz basada en eventos, el *parser* lee el archivo y genera eventos a medida que encuentra elementos, atributos o texto en el archivo, pasando los datos del documento XML directamente a la aplicación (véase Figura 2.11 y Figura 2.12).

inicio documento
inicio elemento: ALUMLIST
inicio elemento: ALUMNO
inicio elemento: NOMBRE
valor: Gloria
fin elemento: NOMBRE
fin elemento: ALUMNO
fin elemento: ALUMLIST
fin documento

Figura 2.11 Una API basada en eventos (toma el ejemplo de datos del documento XML de la Figura 2.8).

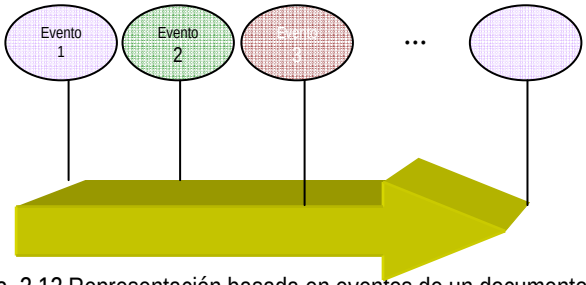

Figura 2.12 Representación basada en eventos de un documento XML.

Tim Bray (uno de los editores de la especificación XML) y David Megginson (desarrollador del parser XML de Microstar) decidieron crear una API basada en eventos para parser XML [15]. La discusión del diseño se publicó en la lista de correo XML-DEV, y mucha gente contribuyó con ideas, comentarios y críticas. Después de un largo periodo de discusiones, la comunidad de XML-DEV desarrolló la primera versión de API Simple para XML: SAX, liberada en mayo de 1998.

SAX es una forma alternativa para analizar documentos XML: contiene en su núcleo tres clases opcionales de ayuda, cinco clases de demostración y tres interfaces que los desarrolladores de un parser SAX deben implementar.

SAX es ideal para procesar archivos de gran tamaño, pero tiene la desventaja de no poder manipular la información una vez procesada, por lo que generalmente se utiliza para leer documentos XML que no serán modificados.

2.6 DTD (*Document Type Definition,* Definición de Tipo de Documento)

Una DTD es una colección de reglas que describen un tipo de documento. El nombre de su elemento raíz debe ser único y es la etiqueta que engloba todo el contenido del documento. Una DTD puede ser usada por muchas instancias del mismo tipo de documento. La definición que identifica una DTD aparece en segundo lugar en un documento XML, después de la especificación que lo identifica como tal, y es opcional; sin embargo, se recomienda definirla cuando se desea que se cumplan ciertas especificaciones y que se incluyan datos requeridos.

La DTD realiza las siguientes tareas:

- Define todos los elementos (nombres de etiquetas) que pueden aparecer en el documento.
- Define el modelo de contenido para cada elemento; es decir, cuales contienen a otros en el caso de etiquetas anidadas, la secuencia y ocurrencia en que dichos elementos deben aparecer.
- Suministra información adicional que puede ser incluida en el documento, como son los Atributos, las Entidades y las Notaciones.
- Aporta comentarios e instrucciones para su procesamiento.

La sintaxis de una DTD es diferente a la sintaxis de los documentos XML, y para hacer una referencia, se ocupa la instrucción <!DOCTYPE>, que enlaza el archivo del documento XML con el archivo DTD.

Declaración de un elemento

La declaración de un elemento se especifica con la palabra <!ELEMENT seguida del nombre del elemento y por su modelo de contenido. La declaración del elemento se termina con el paréntesis angular de cierre (>).

Ejemplo:

```
<!ELEMENT alumnos (alumno)*>
```

Los paréntesis se usan para agrupar elementos dentro del modelo de contenido, como en el siguiente ejemplo:

```
<!ELEMENT alumno (apellido, (nombre-pila | segundo-nombre ),
                  direccion+, foto?)>
```

Un elemento puede contener una lista de elementos, que pude estar formada por alguno de los *tipos de elemento* de la Tabla 2.2.

Tipos de elemento	Descripción
#PCDATA	Se utiliza para analizar datos de caracteres y significa que el elemento puede contener texto. Por lo general, se utiliza para los elementos hoja.
EMPTY	Indica que el elemento es vacío. Siempre indica un elemento hoja.
ANY	Indica que el elemento puede contener cualquier otro elemento declarado en la DTD.

Tabla 2.2 Tipos de elemento.

Ejemplos:

`<!ELEMENT nombre (#PCDATA)>`	Indica que el elemento *nombre* contiene caracteres alfanuméricos.
`<!ELEMENT edad EMPTY>`	Indica que el elemento *edad* no acepta ningún valor.

Los *indicadores de ocurrencia*, indican la cantidad de veces que un elemento se puede repetir, o si es opcional, como se describe en la Tabla 2.3.

Elemento seguido de	Ocurrencia
(ningún indicador)	Puede aparecer una y solo una vez dentro del elemento en el cual fue definido.
+	El elemento se puede repetir, cuando menos una vez.
*	Puede aparecer cero o más veces.
?	El elemento es opcional, y si se incluye no se puede repetir.

Tabla 2.3 Indicadores de ocurrencia.

Ejemplo:

```
<!ELEMENT  alumnos (alumno)*>
```

Indica que el elemento *alumno* puede aparecer cero o más veces dentro del elemento *alumnos*.

```
<!ELEMENT  alumno (nombre+,
direccion, foto?)>
```

Indica que dentro del elemento *alumno*, el *nombre* se puede repetir, el elemento *dirección* solo puede aparecer una vez, y *foto* es opcional.

Los *conectores* separan los hijos en el modelo de contenido e indican el orden en el cual pueden aparecer. En la Tabla 2.4 se definen los conectores que se pueden utilizar.

Conector	Indica que
, (coma)	Los elementos a la derecha y a la izquierda deben aparecer en el mismo orden dentro del documento, también funciona como un AND.
| (barra vertical)	Sólo uno de los elementos de la derecha o de la izquierda de la barra vertical debe aparecer dentro del documento, funciona como un OR.

Tabla 2.4 Conectores.

Ejemplo:

```
<!ELEMENT    alumno    (apellido,
(nombre-pila | segundo-nombre ),
direccion+, foto?)>
```

El elemento alumno se compone de los elementos *apellido*, seguido de *nombre-pila* o *segundo nombre*, seguido de *direccion* y finalmente de *foto*.

Atributos

Los atributos de cada elemento también deben estar declarados en la DTD, con la palabra ATTLIST, y pueden tomar distintos valores, de los cuales se utilizan los siguientes:

Valor	Uso
CDATA	Para atributos de cadena de caracteres.
ID	Para identificador.

Como una opción, la DTD puede especificar un valor predeterminado para el atributo. Si el documento no incluye el atributo, se asume que tiene el valor predeterminado, que puede tomar uno de los cuatro valores siguientes:

Valor	Significado
#REQUIRED	El valor debe ser proporcionado en el documento.
#IMPLIED	Si no se proporciona un valor, la aplicación debe usar su propio valor predeterminado.
#FIXED	Seguido de un valor, significa que el valor del atributo debe ser el valor declarado dentro de la DTD.
Un valor literal	Toma este valor si no se proporciona ninguno en el documento.

DTD externa e interna

La DTD puede definirse dentro o fuera del documento. La declaración <!DOCTYPE siempre está seguida del elemento raíz que aparece en el archivo XML y su contenido es lo que la identifica como interna o externa.

```
<!DOCTYPE elementoRaizXml
```

Una DTD es *externa* cuando en el documento XML se hace una referencia a la ubicación y al archivo que contienen la DTD a utilizar; en este caso el elemento *raíz* de la cláusula *DOCTYPE* debe estar seguido de la palabra *SYSTEM,* cuando las etiquetas que componen la DTD no son públicas, o por la palabra *PUBLIC,* cuando ya se han publicado. Posteriormente, se coloca entre comillas el valor de la URL que especifica la ubicación de la DTD y se cierra la etiqueta con el símbolo “>”.

Ejemplo:

```
<!DOCTYPE elementoRaizXML SYSTEM
"Http://148.204.20.13:8080/Documento.dtd">
```

Una DTD que se incluye dentro del documento XML se denomina *interna*. En ella, el elemento raíz de la cláusula *DOCTYPE* de un documento XML precede a un paréntesis cuadrado que abre, seguido del subconjunto de declaraciones que componen a la DTD, finalizando con un paréntesis cuadrado que cierra y el angular "**>**" que delimita la declaración <!DOCTYPE.

Ejemplo:

```
<?xml version = "1.0"?>
<!DOCTYPE elementoRaizXML [
      <!ELEMENT elementoRaizXML(elemento1+, elemento2+, elemento3+,
      elemento4?)
      <!ELEMENT elemento1(#PCDATA)>
      <!ATTLIST elemento1 atributo1 (valor1 | valor2) #IMPLIED>
      <!ELEMENT elemento2 (#PCDATA)>
      <!ELEMENT elemento3 (#PCDATA)>
      <!ELEMENT elemento4 EMPTY>
       ]>
<elementoRaizXML>
  <elemento1 atributo1="valor">Contenido del elemento1</elemento1>
  <elemento2>Contenido del elemento2</elemento2>
  <elemento3>Contenido del elemento3</elemento3>
  <elemento4/>
</elementoRaizXML>
```

2.7 Reglas de XML

La recomendación actual de XML contiene varias reglas, en total ochenta y ocho [25]. A continuación, (véase Tabla 2.5) se muestran: el número de regla, su descripción, y algunos ejemplos de las que fueron utilizadas en el Módulo XML para acceder al SABD SQLmx.

Núm. de Regla	Descripción	Explicación y/o ejemplo

1	Define la estructura de un documento XML.	Un documento XML se compone de un prólogo, elementos y comentarios e instrucciones de procesamiento opcionales.
2	Hace referencia a un carácter.	Acepta cualquier carácter UNICODE.
3	Define espacio en blanco.	Un espacio en blanco puede ser: un retorno de carro, una tabulación o un espacio vacío.
4 5 6	Definición de nombres.	Un nombre se compone de una letra inicial, seguida de más letras o dígitos, o guión bajo (_) o guión (-) o dos puntos (:).
15	Comentarios.	Representan elementos que no se van a procesar. Ejemplo: `<!--  Consulta: "select * from parte"  -->`

22	Define el prólogo del documento.	Se compone de la declaración *XML* y la declaración *DOCTYPE*.
		Ejemplo:
		`<?xml version = "1.0" encoding = "ISO-8859-1" standalone = "yes" ?>`
		`<!DOCTYPE raiz SYSTEM "http://148.204.20.13:8080/datos/archivo.dtd">`

23	Declaración XML.	Identifica a un documento XML como tal. Ejemplo: `<?xml version = "1.0" encoding = "ISO-8859-1" standalone = "yes" ?>`
24	Información de la versión utilizada en el documento.	La palabra *version* especifica el número de versión de la recomendación XML a la que se apega el documento y debe estar seguida por un signo de igual y el número que la identifica entre comillas o apóstrofes. Ejemplo: `version = "1.0"`
25	Declaración de igualdad.	El signo de igual debe ir entre dos espacios en blanco. Ejemplo: `version = "1.0"`
26	Número de versión.	El número de la versión se compone de cualquiera de los caracteres (0-9 a-z A-Z_.:) ó guión (-). Ejemplo: `1.0`
27	Miscelánea.	Otros elementos que pueden aparecer en un documento XML tales como comentarios, instrucciones de procesamiento o espacios en blanco.
28	Declaración *DOCTYPE*.	Especifica la DTD que define los elementos que componen el documento XML. Ejemplo: `<!DOCTYPE raiz SYSTEM "http://148.204.20.13:8080/datos/archivo.dtd">`
29	Declaración de marcas.	Especifica la composición de una etiqueta XML. Una etiqueta puede ser, de inicio, de fin o vacía.

32	Declaración *standalone.*	Indica si el documento es independiente de otros. Para ello se utiliza la palabra *standalone* y un valor de verdad (yes o not). Ejemplo: `standalone = "yes"`
39	Definición de un elemento.	Define la estructura lógica de un elemento mediante la palabra *ELEMENT*. Ejemplo: `<!ELEMENT registro (NoPar?,NomPar?,ColorPar?)>`
40	Etiqueta de inicio.	Define el inicio de un elemento XML. Ejemplo: `<registro numero = "id-1">`
41	Atributos.	Los atributos representan características que describen al elemento de la etiqueta a la que corresponden. Ejemplo: atributo *número* del elemento *registro*, del ejemplo anterior `numero = "id-1"`
42	Etiqueta de fin.	Delimita el fin de un elemento. Ejemplo: `</registro>`
43	Contenido de un elemento.	Especifica el orden y contenido de un elemento. Ejemplo: `resultset (registro)*`
44	Etiqueta para elementos vacíos.	Representa elementos con valor desconocido. Ejemplo: `<ColorPar/>`
45	Declaración de tipo de elemento.	El tipo de elemento se especifica por su contenido. Ejemplo: `<!ELEMENT registro (NoPar?,NomPar?,ColorPar?)>`
47	Modelo de contenido (hijos) de un elemento.	Un elemento puede o no tener hijos. Del ejemplo anterior, los hijos son: `(NoPar?,NomPar?,ColorPar?)`
48	Ocurrencia de un elemento.	Indica la frecuencia de aparición de un elemento. Ejemplo: `NoPar?` opcional `(registro)*` cero o más
49	Opciones de contenido.	Detalla entre paréntesis las opciones que se pueden elegir para introducir un contenido a un elemento. Ejemplo: `(insert|update|delete)*`

50	Secuencia de contenido.	Indica el orden en que aparecen los elementos que componen a otro. Ejemplo: `(NoPar?,NomPar?,ColorPar?)`
51	Declaración de contenido mixto: texto y/o elementos.	Son elementos que contienen datos carácter y elementos hijos. La palabra reservada #PCDATA indica que los elementos están formados de caracteres y que deben ser analizados para revisar si hacen referencia a entidades.
52	Lista de atributos.	Los atributos que forman un elemento se declaran con la palabra ATTLIST. Ejemplo: `<!ATTLIST NomPar tipo CDATA #IMPLIED>`
53	Definición de atributo.	Un atributo se define indicando su tipo y su declaración por defecto para su inclusión en un elemento. Ejemplo: `NomPar tipo CDATA #IMPLIED`
54	Tipos de atributos.	Define los tipos de atributos permitidos. Para el ejemplo anterior **CDATA** representa el tipo.
55	Tipos de atributo cadena.	Permite escribir una secuencia de caracteres mediante la palabra CDATA.
60	Valor por defecto de presencia de un atributo	Indica si se requiere el valor de un atributo, y si no existe, como debe tratarse al procesarlo. Para el siguiente ejemplo: #IMPLIED especifica que el atributo *tipo* del elemento NomPar es opcional. `NomPar tipo CDATA #IMPLIED`

Tabla 2.5 Reglas de XML.

De las reglas anteriores, las utilizadas en el presente trabajo comprenden aquellas que conforman los documentos XML y las estructuras lógicas que los componen. No se incluyen las reglas que declaran secciones CDATA, las cuales son útiles para contener información que no se desea procesar. Otras reglas, como las que definen atributos con valores enumerados, no se consideran, ya que en esta aplicación el usuario proporciona el documento XML, y la definición de los elementos que lo componen se genera en tiempo de ejecución; esto implica que si el usuario decide no incluir un atributo, la definición del mismo le asignará un valor por defecto, que altera lo que se desea obtener. Por ejemplo, si se tiene un atributo de condición enumerado que debe tomar uno de tres valores (>, < ó =), con el valor = como predeterminado, implica que cuando se genere la definición del documento a recibir, si no se incluye un valor para ese atributo, se le asignará el valor =, pero tal vez esto no sea lo que quiere el usuario; la aplicación al no encontrar el atributo lo agregará y hará la comparación respectiva, obteniendo resultados diferentes a los deseados.

El resto de las reglas de la recomendación XML no incluidas hacen referencia a estructuras físicas, las que definen entidades (una entidad puede hacer referencia a

datos binarios, por ejemplo imágenes y archivos ejecutables, los cuales no se aplican en SABD relacionales).

2.8 El Lenguaje Estructurado de Consulta y su utilización en la aplicación.

SQL es un lenguaje de consulta para organizar, administrar y recuperar datos almacenados en una base de datos relacional y que sirve para comunicarse con el SABD.

ANSI (*American National Standards Institute*, Instituto Nacional Americano de Normalización) e ISO (*International Standards Organization*, Organización Internacional para la Estandarización), publicaron un estándar oficial de SQL en 1986. Para mejorar algunos aspectos del estándar original, el comité ANSI elaboró uno nuevo, el cual fue aprobado en 1992 y reconocido como el SQL92 ó SQL2.

En este trabajo se utiliza el estándar SQL92, del cual se consideran los siguientes aspectos:

- La utilización de sentencias de manipulación de datos: SELECT, DELETE, UPDATE e INSERT.

- Al ejecutar una sentencia SELECT (sin incluir subconsultas), se obtiene un resultado que es transformado a XML y que se transfiere al usuario.

Consulta en SQL	Resultado expresado en XML

`Select * from parte`	`<resultset>`
	`  <registro numero="id-1">`
	`  <NoPar tipo="INTEGER">1</NoPar>`
	`  <NomPar tipo="CHAR">CHASIS</NomPar>`
	`  <ColorPar tipo="VARCHAR">NEGRO</ColorPar>`
	`  </registro>`
	`  <registro numero="id-2">`
	`  <NoPar>2</NoPar>`
	`  <NomPar>MOTOR</NomPar>`
	`  <ColorPar>NEUTRO</ColorPar>`
	`  </registro>`
	`</resultset>`

- La sentencia INSERT ha sido implementada siguiendo el SQL92, por lo que permite recibir todos o algunos de los valores de los campos especificados en un esquema.

  Por ejemplo, para la tabla PARTE cuyo esquema es: *PARTE (NumPar, NomPar , ColorPar),* se desea insertar 2 registros de los cuales se desconoce el atributo *ColorPar*.

  El documento XML envía los registros etiquetados de la siguiente manera:

Sentencias SQL para cada uno de los registros a insertar	Documento XML equivalente que contiene los valores a insertar
insert into parte (NumPar,NomPar) **values** (1,'LLANTA'); **insert into** parte (NumPar,NomPar) **values** (2,'MOTOR');	<parte> <insert> <NumPar>1</NumPar> <NomPar>LLANTA</NomPar> </insert> <insert> <NumPar>2</NumPar> <NomPar>MOTOR</NomPar> </insert> </parte>

– En el caso de las sentencias DELETE y UPDATE, las condiciones WHERE pueden incluir los conectores AND y OR, e incluir paréntesis para indicar la precedencia.

Por ejemplo: si se desean borrar los registros de la tabla PARTE cuando el color sea "*rojo*" o "*negro*" y el nombre sea "*motor*" o "*llanta*".

Sentencia expresada en SQL	Representación equivalente en XML
delete from parte **where** (ColorPar="ROJO" **or** ColorPar="NEGRO") **and** (NomPar="MOTOR" **or** NomPar="LLANTA")	<parte> <delete> <parentesis con="and"> <parentesis con="or"> <sentencia comp="="> <ColorPar>ROJO</ColorPar> </sentencia> <sentencia comp="="> <ColorPar>NEGRO</ColorPar> </sentencia> </parentesis> <parentesis con="or"> <sentencia comp="="> <NomPar>MOTOR</NomPar> </sentencia> <sentencia comp="="> <NomPar>LLANTA</NomPar> </sentencia> </parentesis> </parentesis> </delete> </parte>

– En la modificación de datos (usando la cláusula UPDATE), pueden agregarse las operaciones aritméticas de suma, resta, multiplicación y división.

Por ejemplo, si se desea aumentar el salario a todos los empleados en un 5 %, la actualización a la tabla se da de la siguiente manera:

Sentencia en SQL	Representación equivalente en XML
update empleado	<empleado> <update>

set salario = salario *1.05	<set> <condicion fn="*"> <salario>1.05</salario> </condicion> </set> <update> </empleado>

Realizar operaciones con documentos XML es de mayor utilidad en el caso de consultas e inserciones, ya que ambas envían o reciben gran cantidad de datos. XML facilita al usuario la transferencia de muchos registros a la vez, evitando que lo haga sentencia por sentencia (en el caso de la inserción), y para las consultas, retorna todos los registros del resultado, etiquetados de tal forma que puedan procesarse posteriormente si se desea; debido a esto, no se incluyeron las sentencias de definición de datos.

2.9 Resumen

En este capítulo se ha definido el marco teórico en que se basa el desarrollo de la tesis, se explican las características principales de XML, SQLmx y DTD, de las que se puede resumir que XML es un metalenguaje de marcas, que permite estructurar documentos válidos si va aunado con una DTD en la que se definan los elementos que lo componen.

Se logra establecer que el SABD relacional SQLmx extiende su capacidad de respuesta a peticiones hechas en Internet, adoptando la tecnología de vanguardia: XML, con la que da respuesta a peticiones de manipulación de datos.

Capítulo 3

Arquitectura del Módulo XML

3.1 Introducción

En este capítulo se explica la arquitectura del Módulo XML, su análisis y diseño están desarrollados en UML (*Unified Model Language,* Lenguaje Unificado de Modelado), por lo que se esquematizan con diagramas de casos de uso, de secuencia y de clases. Asimismo, se presentan los diagramas funcionales del sistema.

3.2 Arquitectura

El Módulo XML está diseñado para dar soporte a las sentencias de manipulación de datos del SABD SQLmx, por lo que se divide en dos submódulos: el de consulta, que permite hacer indagaciones a los datos almacenados, y el de modificación de datos, que permite insertar, borrar o actualizar registros de una tabla.

Módulo de consulta a la base de datos

Este módulo representa la forma en que el cliente puede consultar la base de datos desde Internet, para recibir un archivo XML con los resultados de la misma.

Utilizando el Lenguaje Estructurado de Consultas SQL, se introduce a través de una página HTML una sentencia de consulta de tipo *select * from tabla,* que mediante el protocolo HTTP se envía al servidor Web utilizando un *servlet*. Haciendo uso de JDBC, el *servlet* obtiene el resultado del motor SQLmx, que es el encargado de validar la sintaxis de la consulta realizada.

Una vez devueltos los resultados, el *servlet* los transmite al Módulo XML, que estructura un documento DTD con los *metadatos* proporcionados por el JDBC, y genera con el resultado una estructura basada en la interfaz DOM, la cual sirve para construir un archivo con formato XML que contiene los datos. Una vez obtenidos estos dos documentos, se integran y se envían al analizador XML que verifica la validez del archivo XML, con la DTD que especifica el modelo de contenido de los datos que incluye; si está bien formado y si es válido, se regresa al *servlet,* que da respuesta a la petición.

El documento XML que llega al cliente incluye una DTD interna. Finalmente, el usuario puede dar formato a través de una aplicación XSL (*eXtensible Stylesheet Language*, Lenguaje de Estilo Extensible) a los datos contenidos en el documento que recibe (véase Figura 3.1)

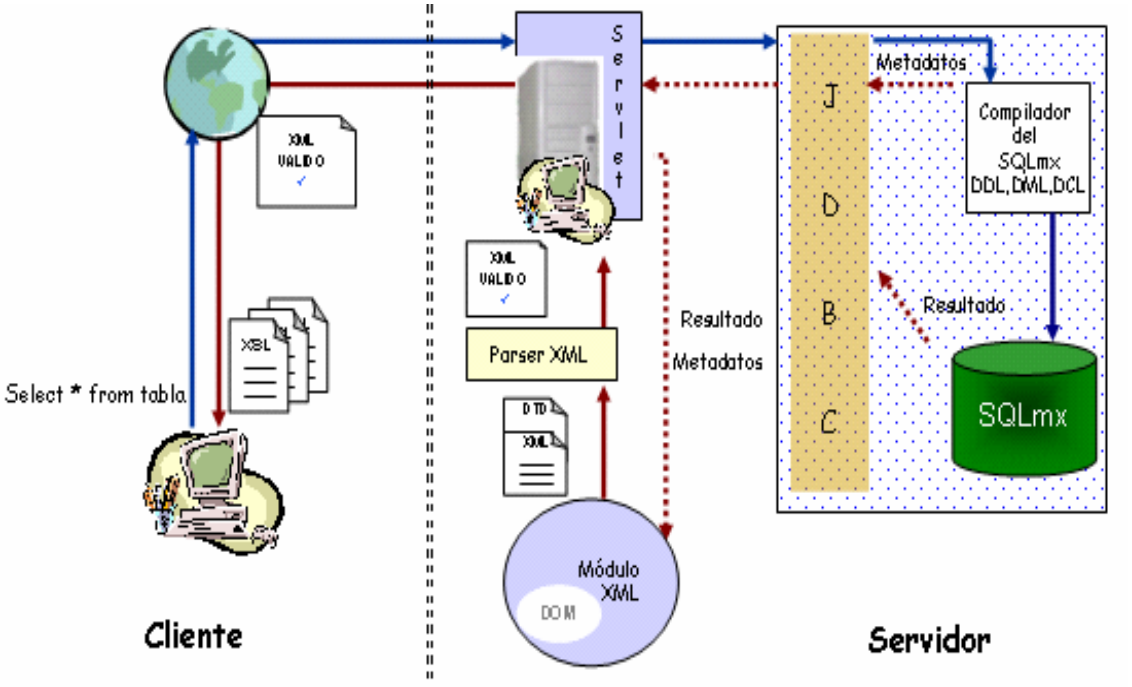

Figura 3.1 Módulo de consulta.

Módulo de modificación a la base de datos.

Una base de datos puede sufrir actualizaciones como son: inserción, borrado de datos, o cambio del contenido de un registro. Es posible insertar un conjunto de registros contenido en un documento XML, es decir, en un documento puede enviarse la cantidad de registros que se deseen introducir o actualizar. Utilizando un correcto etiquetado que cumpla con el esquema de la base de datos, el documento XML puede ser transformado e insertar tantos registros como haya en el archivo XML. Otra opción, es enviar en el documento XML las etiquetas que incluyan los valores para hacer eliminaciones.

Para ello, el cliente envía un documento XML con toda la información a guardar, modificar o borrar en la base de datos; el *servlet* obtiene dicho documento del lado del servidor y lo envía al Módulo XML, que se encarga de revisar el catálogo del sistema para recuperar el esquema correspondiente a la tabla que se va a modificar, generando su respectiva DTD; a partir de aquí, mediante DOM, se construye un árbol

si el documento XML recibido es válido; esto es, que cumpla con el esquema de la tabla a modificar. De dicho árbol se descompone el documento XML en sentencias *insert*, *delete* o *update*, las cuales se transmiten al motor del SABD SQLmx para introducirlas en la tabla correspondiente (Véase Figura 3.2)

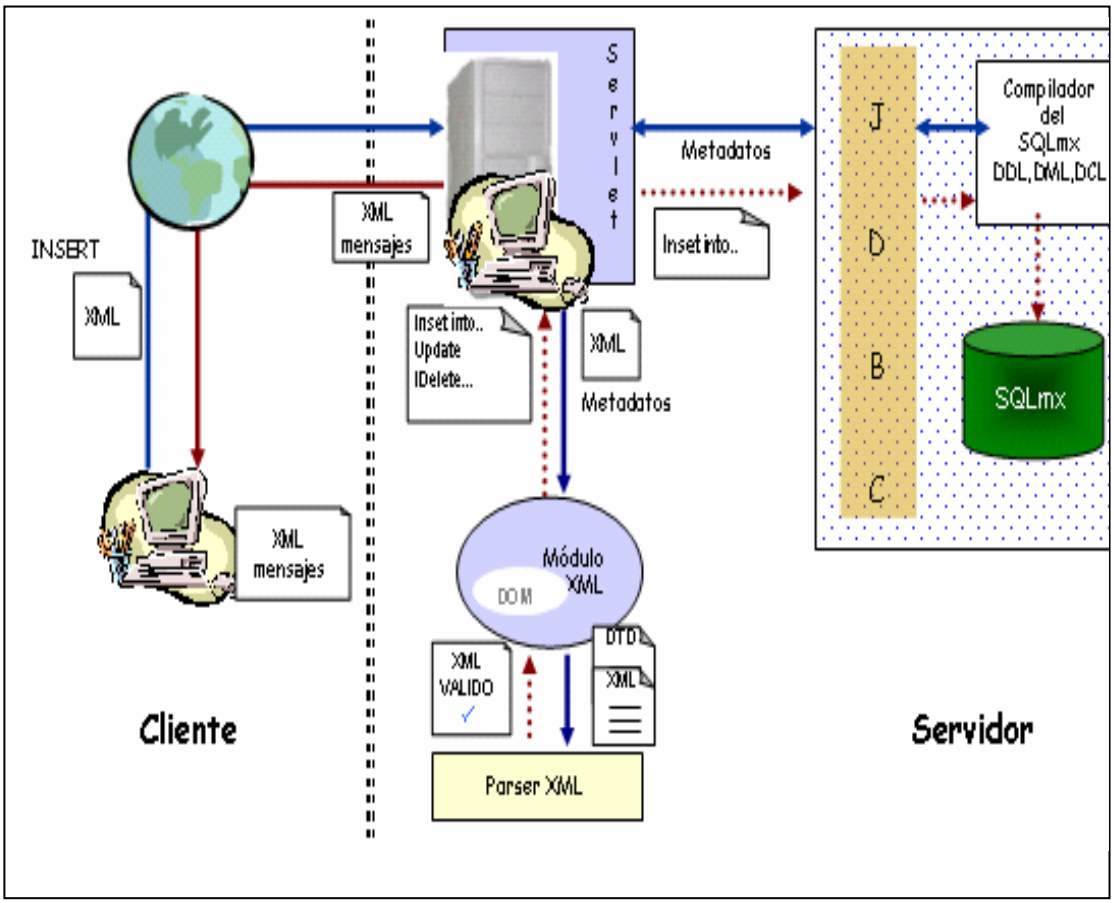

Figura 3.2 Módulo de modificación de la base de datos.

3.3 Análisis y diseño del Módulo XML

Cuando un usuario de bases de datos realiza una petición de consulta o modificación, se genera un documento DTD, y para el caso de consultas, también un documento XML validando ambos documentos. Los datos que estos documentos contienen se extraen de una base de datos SQLmx. El diagrama del modelado del sistema utilizando casos de uso se muestra en la Figura 3.3.

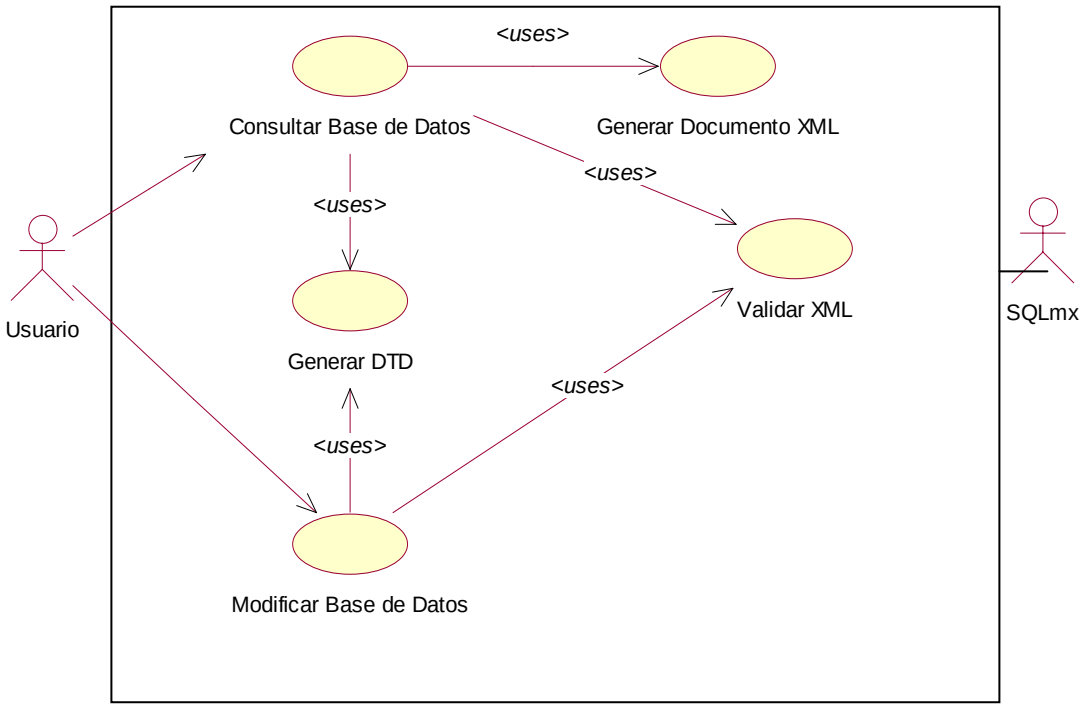

Figura 3.3 Caso de uso de primer nivel del Módulo XML.

El modelado anterior muestra dos casos de uso principales: Consultar base de datos y Modificar base de datos.

3.3.1 Caso de uso Consultar base de datos

El caso de uso Consultar Base de Datos surge de la descomposición del caso de uso de primer nivel del Módulo XML (véase Figura 3.4).

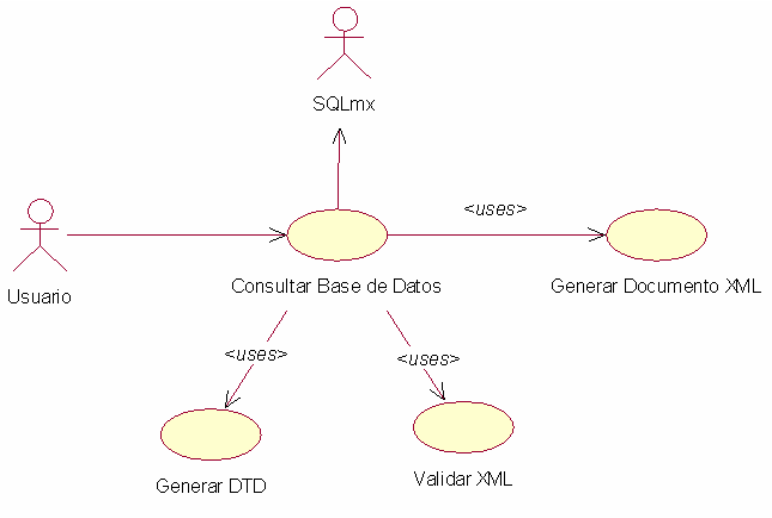

Figura 3.4 Caso de uso Consultar Base de Datos

Lista de eventos para el caso de uso Consultar Base de Datos

Los eventos que se generan cuando el usuario consulta la base de datos, desde que activa el proceso hasta cuando finaliza, son:

1. El usuario envía una sentencia SQL al módulo a través del *servlet*.
2. Se busca el controlador de conexión a la base de datos.
3. Se obtiene conexión a la base de datos usando JDBC.
4. El *servlet* pide ejecutar la consulta.
5. SQLmx responde a la consulta.
6. El *servlet* recupera los datos y *metadatos* del resultado.
7. Se genera la DTD correspondiente.
8. El *servlet* invoca la creación de un documento XML.
9. Se construye el árbol DOM para ese documento.
10. El árbol DOM se etiqueta para formar el documento XML.
11. El *servlet* recupera el documento XML.
12. El *servlet* regresa al visualizador el documento XML resultado.

3.3.2 Diagrama de secuencia Consultar base de datos.

El diagrama de secuencia muestra el orden que se sigue en el proceso de consulta a la base de datos y los objetos principales que lo componen. La Figura 3.5 realiza en la línea de tiempo los eventos que se detallan para el caso de uso correspondiente.

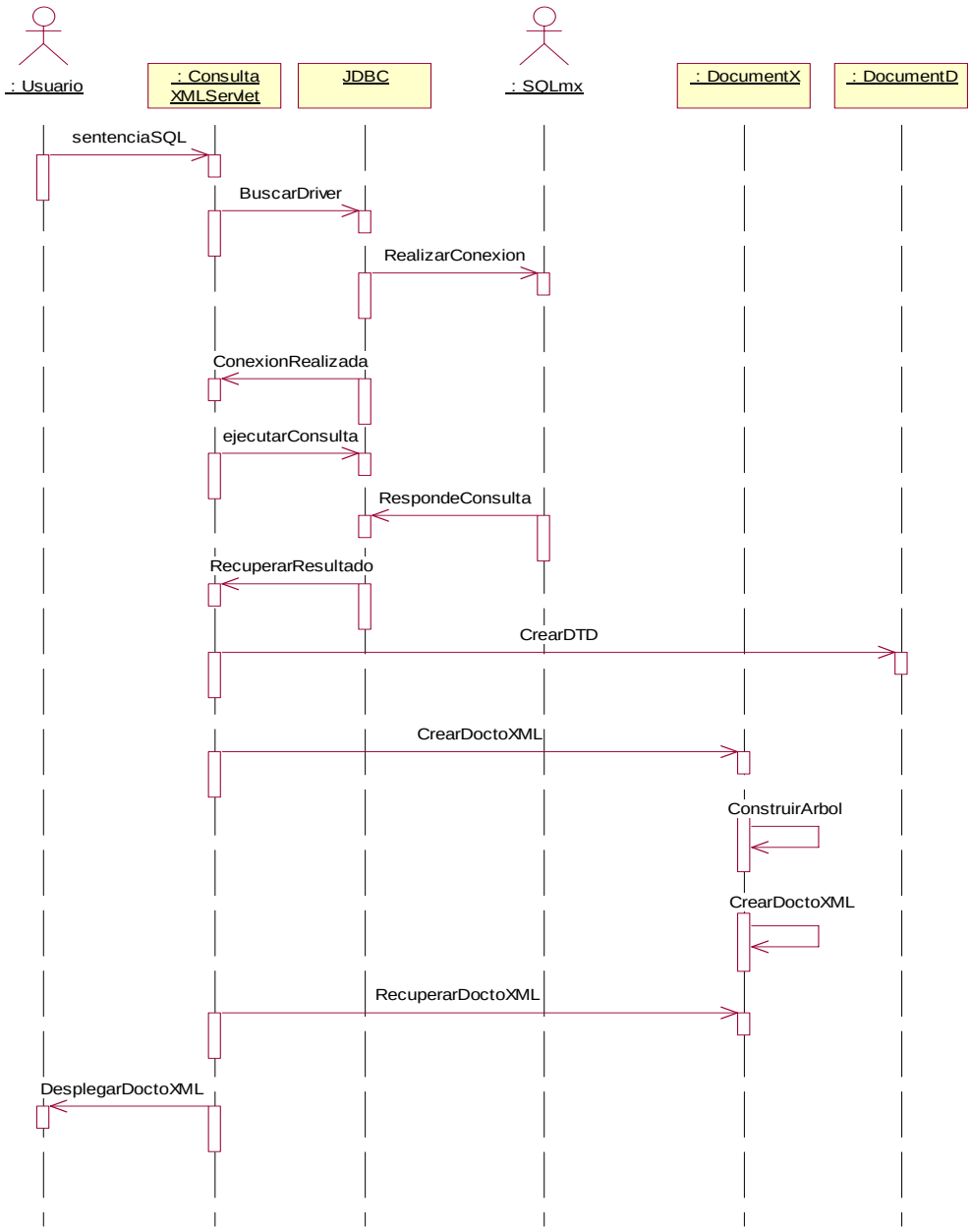

Figura 3.5 Diagrama de secuencia de Consultar base de datos.

3.3.3 Diagrama de clases para Consultar base de datos

El diagrama de la Figura 3.6 muestra las clases y su relación para Consultar la base de datos.

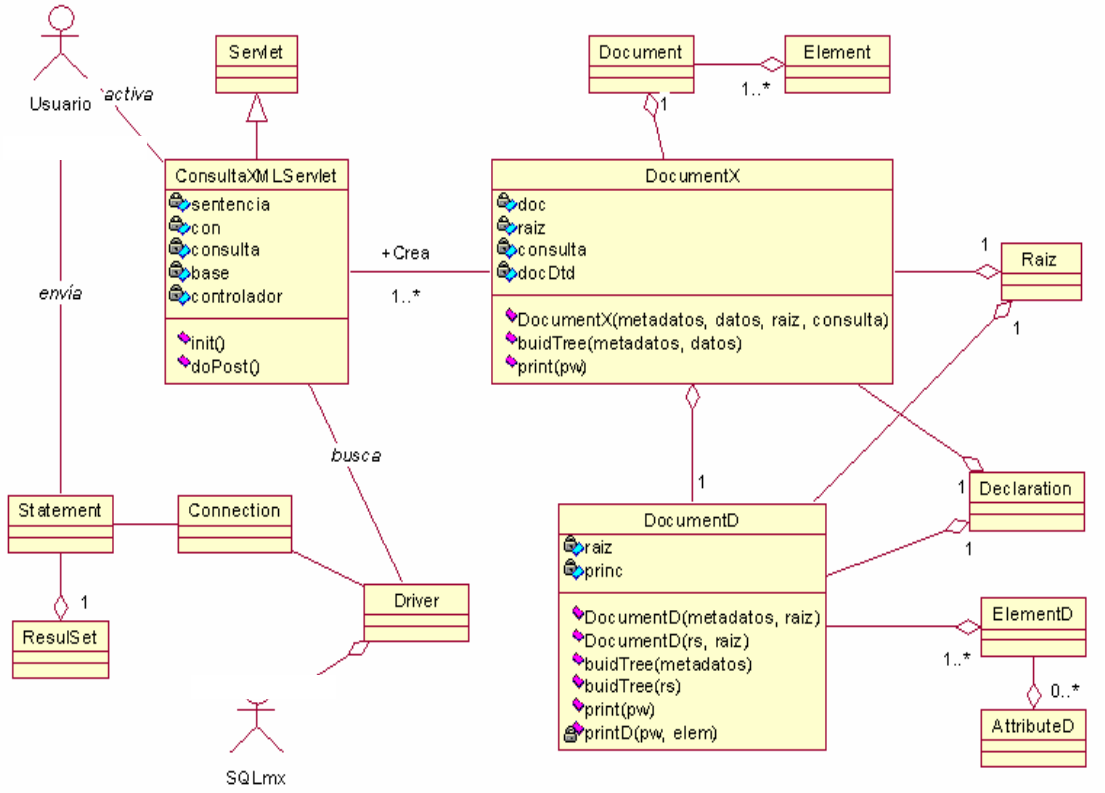

Figura 3.6 Diagrama de clases para Consultar base de datos.

El *servlet* de tipo *ConsultaXMLServlet* se activa cuando el usuario envía la sentencia SQL a través de Internet utilizando HTTP (Véanse Figuras 3.5, 3.6 y 3.7).

El *servlet* sobrescribe el método *doPost()* que hereda de la clase *Servlet*, instanciando clases de JDBC, para buscar la clase que implementa el *driver* que controla a SQLmx; posteriormente crea la conexión mediante un objeto tipo *Connection*, con la que llama a una instancia de la clase *Statement* para ejecutar la sentencia del usuario. El resultado contenido en un objeto *ResultSet* es devuelto al *servlet*. El *servlet* genera un documento XML mediante la clase *DocumentX* (Véanse Figuras 3.5, 3.6 y 3.8).

DocumentX es una clase que se compone de elementos DOM, de un objeto *Raíz*, un objeto de tipo *Declaration* y un documento DTD para validarse. La DTD se crea con la clase *DocumentD*, que para definir cada etiqueta XML, se compone de elementos de tipo *ElementD*, los cuales pueden tener 0 ó más atributos, como se observa en el diagrama de clases (Véanse Figuras 3.5, 3.6 y 3.9).

Finalmente, generado el documento XML por el método *print()* de la clase *DocumentX*, el *servlet* lo envía al visualizador para que lo despliegue en pantalla (Véanse Figuras 3.5, 3.6 y 3.10).

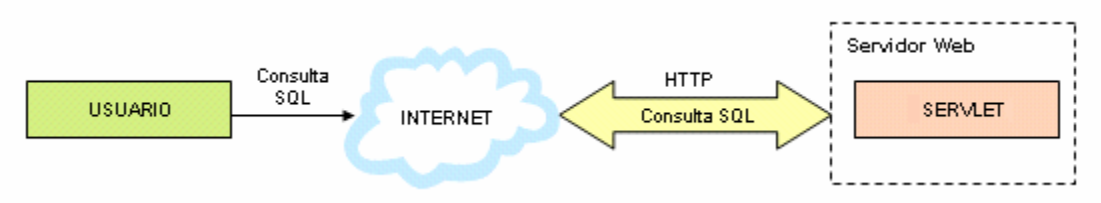

Figura 3.7 Invocación del *servlet* para consultar la base de datos.

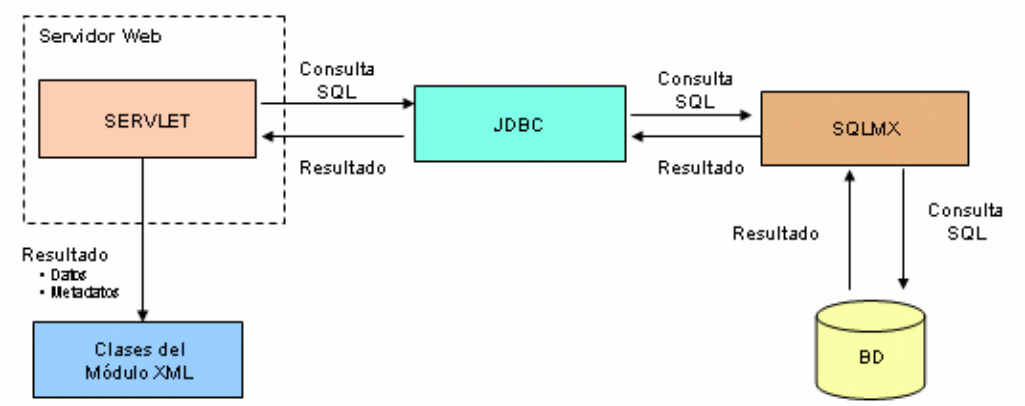

Figura 3.8 Ejecución de la consulta.

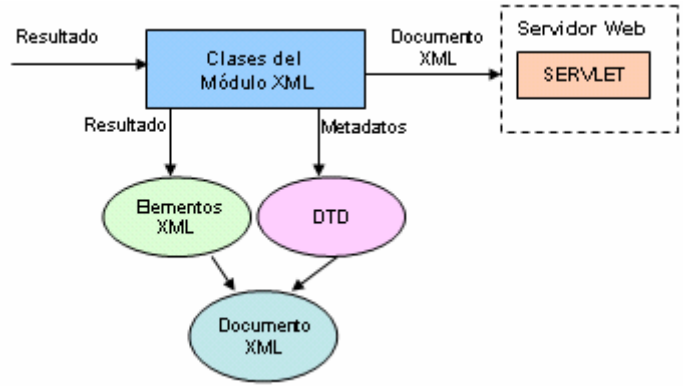

Figura 3.9 Creación del documento XML.

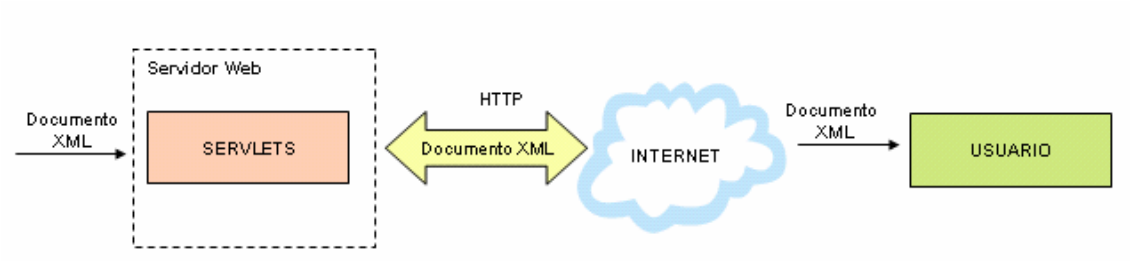

Figura 3.10 Respuesta a la consulta.

3.3.4 Caso de uso Modificar base de datos

Lista de eventos para el caso de uso Consultar base de datos

Los eventos que se muestran en la Figura 3.11, se generan cuando el usuario solicita algún cambio en la base de datos, ya sea para insertar, borrar o actualizar**:**

1. El usuario envía un documento XML que contiene los datos a procesar, el nombre de la tabla, el nombre de la base de datos a la que pertenece, su ubicación, el controlador del SABD y su clave de acceso.

2. El *servlet* solicita separar los parámetros.

3. El *servlet* valida los parámetros.

4. Busca el controlador JDBC para el SQLmx.

5. Se conecta con la base de datos mediante JDBC.

6. El *servlet* consulta el diccionario de datos para generar los *metadatos* de la tabla.

7. El *servlet* genera la DTD.

8. Se valida el documento XML recibido.

9. Se descompone en sentencias de acuerdo al proceso a realizar: inserción, borrado o modificación.

10. El *servlet* extrae las sentencias SQL generadas.

11. El *servlet* ejecuta en SQLmx cada una de las sentencias recibidas.

12. Se devuelve al usuario un mensaje del estado de la base de datos administrada por SQLmx.

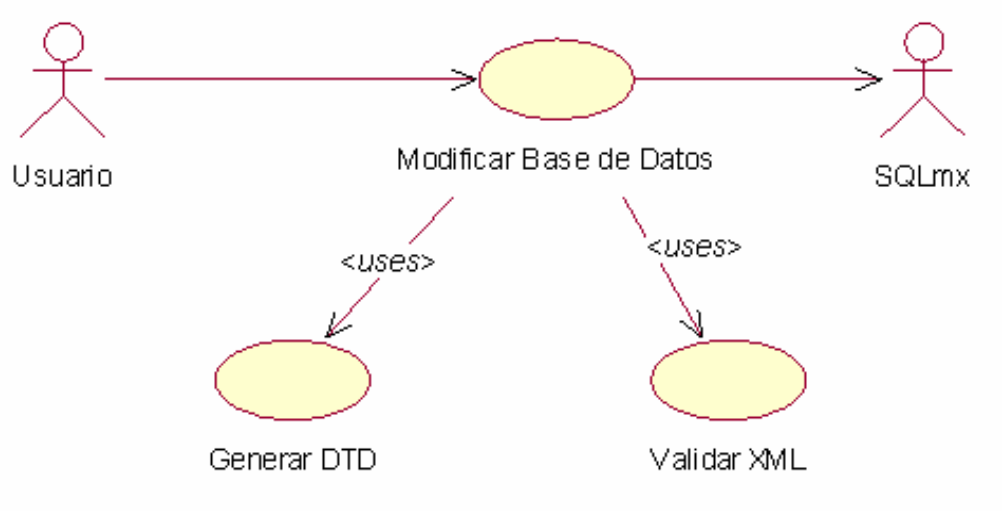

Figura 3.11 Caso de uso Modificar base de datos.

3.3.5 Diagrama de secuencia Modificar base de datos.

El diagrama de la Figura 3.12 muestra la secuencia que se sigue en el proceso de Modificación de los registros de la base de datos y muestra los objetos principales que lo componen.

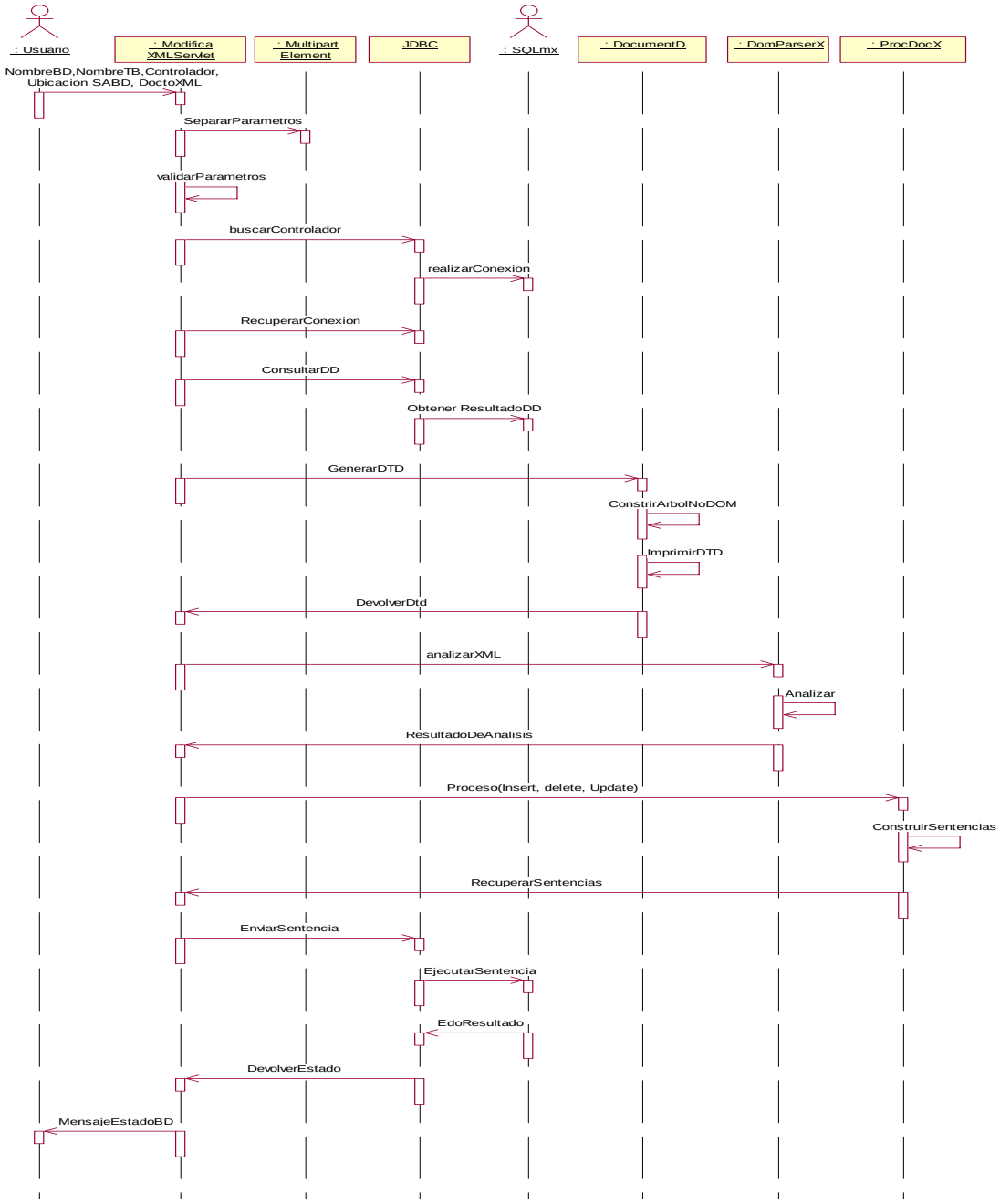

Figura 3.12 Diagrama de secuencia de Modificar base de datos.

3.3.6 Diagrama de clases para Modificar base de datos

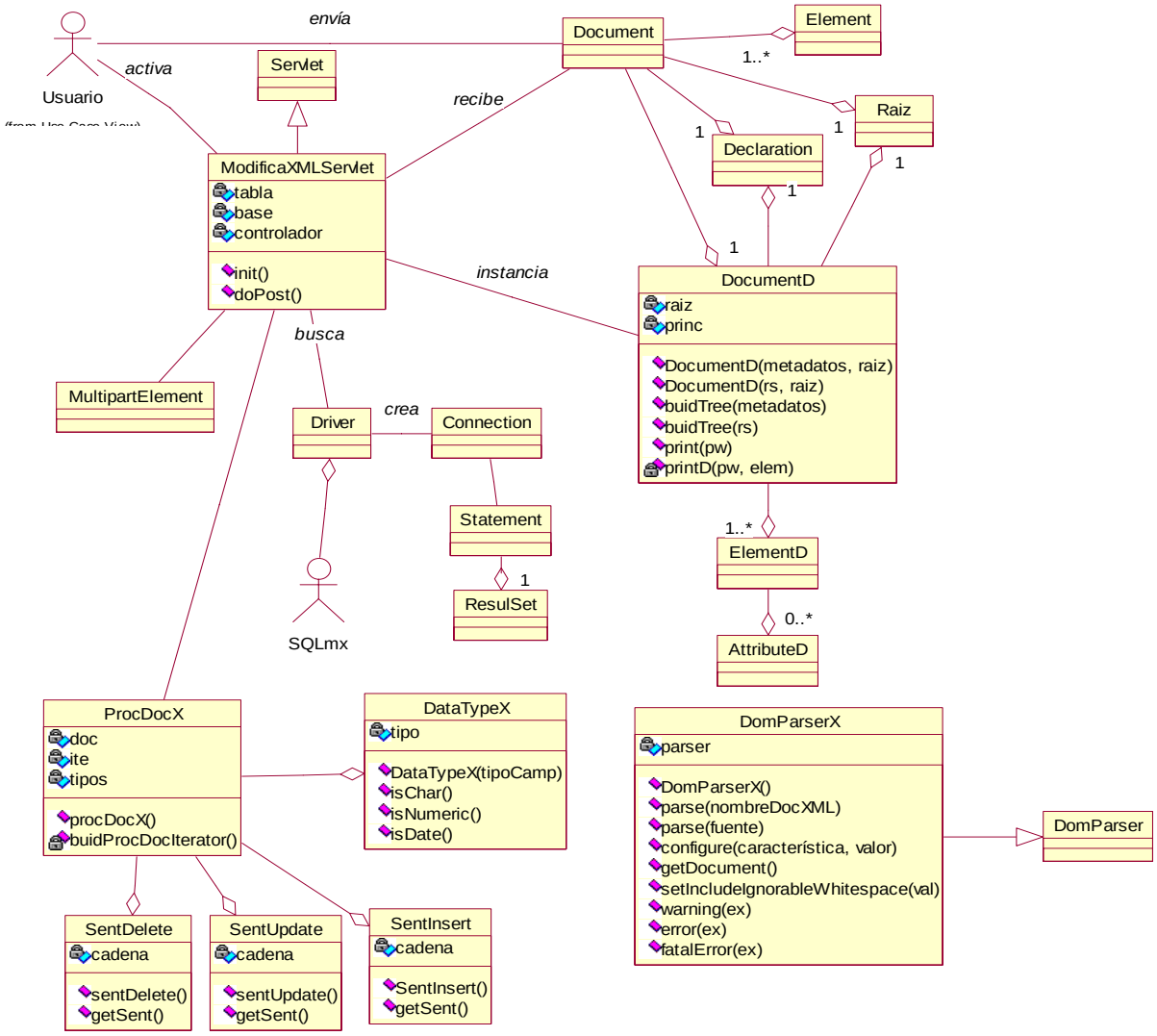

Figura 3.13 Diagrama de clases para Modificar base de datos.

El *servlet* de tipo *ModificaXMLServlet* se activa cuando el usuario envía el nombre de la Base de Datos, el nombre de la tabla a afectar, el controlador y la dirección del SABD y el documento XML que contiene los datos y/o registros (véanse Figuras 3.12, 3.13 y 3.14).

Este *servlet* sobrescribe el método *doPost()* que hereda de la clase *Servlet* y obtiene los datos enviados por el cliente, descomponiendo el flujo en los elementos respectivos, usando la clase *MultipartElement.* En caso de ser correctos, utiliza clases de JDBC para buscar la clase que implementa al controlador que maneja al SQLmx y

posteriormente crea la conexión mediante el objeto *Connection,* con la que llama a una instancia de *Statement* para buscar en el diccionario de datos, la existencia de la tabla y recuperar sus *metadatos* (véanse Figuras 3.12, 3.13 y 3.15).

Con los *metadatos*, se genera la DTD a través de la clase *DocumentD,* para ser devuelta al *servlet*. El *servlet* pide a la clase *DomParserX* la validación del documento enviado por el usuario y la DTD correspondiente (véanse Figuras 3.12, 3.13 y 3.16).

Un objeto de la clase *ProcDocX* realiza la descomposición del documento XML en las sentencias SQL *insert*, *delete* o *update* (véanse Figuras 3.12, 3.13 y 3.17).

El *servlet* recupera las sentencias de *ProcDocX*, y las pasa una a una al JDBC para que las ejecute en el SQLmx (véanse Figuras 3.12, 3.13 y 3.18).

Finalmente, se regresa un mensaje al usuario con el estado de la base de datos resultante respecto al documento recibido (véanse Figuras 3.12, 3.13 y 3.19).

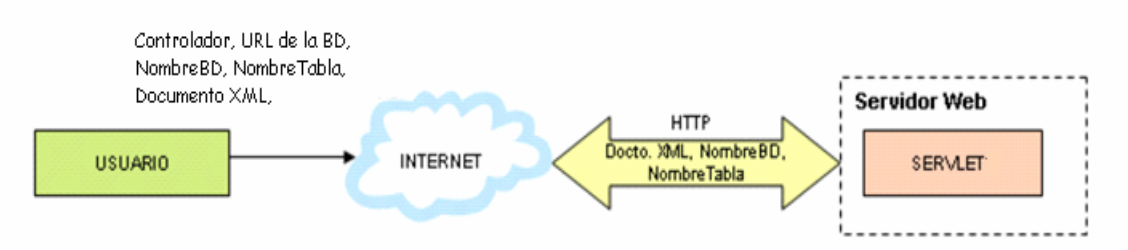

Figura 3.14 Envío de un documento XML para modificar la base de datos.

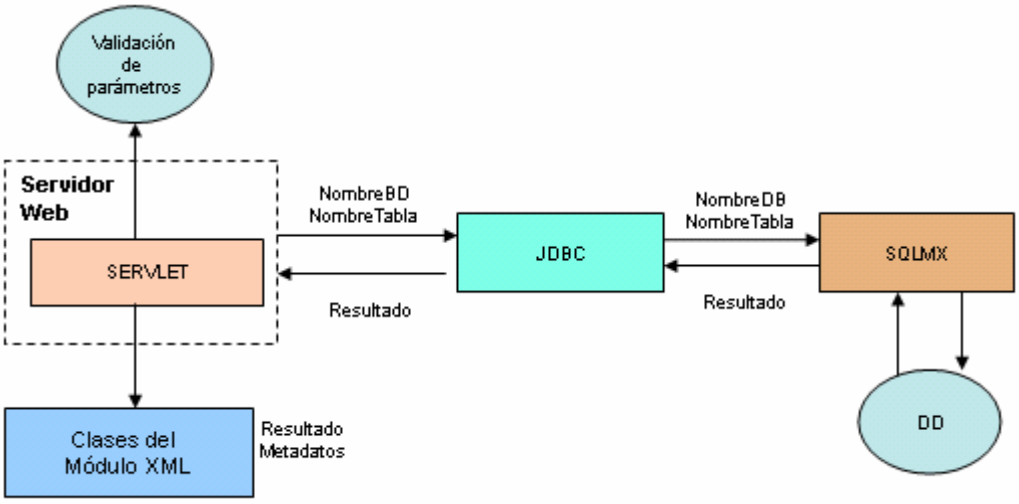

Figura 3.15 Validación de parámetros.

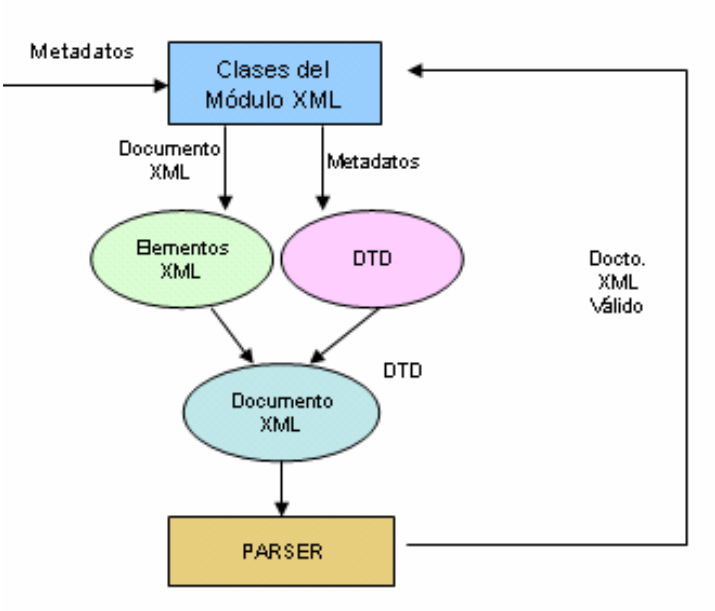

Figura 3.16 Generación de DTD y validación del documento XML recibido.

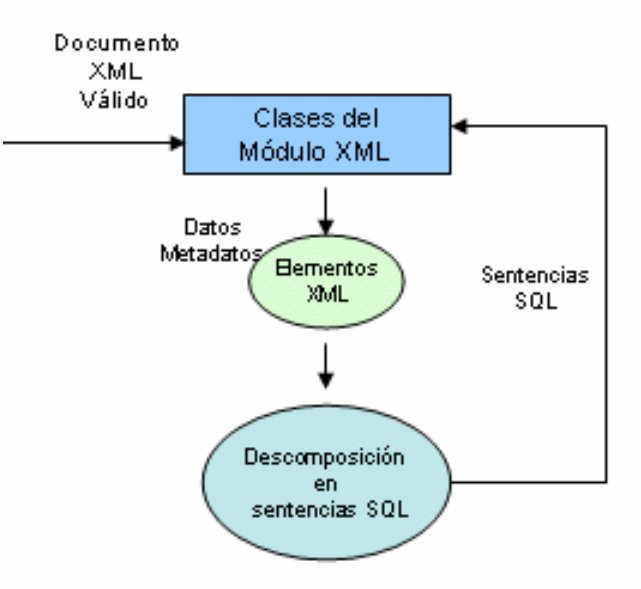

Figura 3.17 Descomposición del documento XML en sentencias SQL.

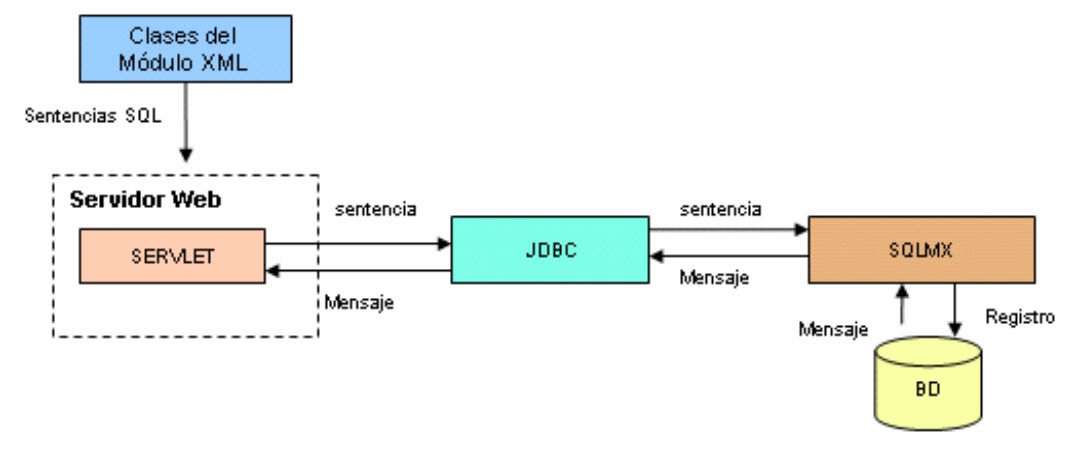

Figura 3.18 Ejecución por lotes de las sentencias SQL.

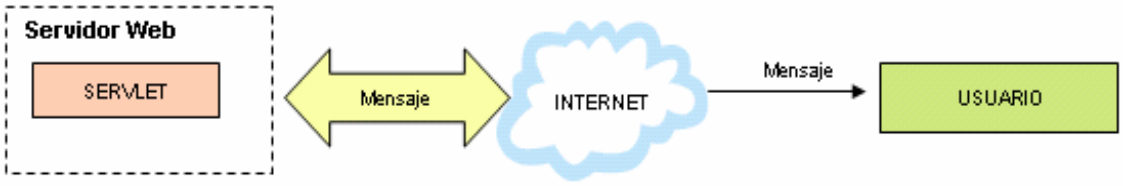

Figura 3.19 Despliegue de mensaje de actualización o resultado al usuario.

3.4 Puntos clave del diseño

El diseño del Módulo XML toma en cuenta los siguientes puntos:

- ***El DML (Data Management Language*, Lenguaje de Manipulación de Datos*) del estándar SQL-92***. XML es utilizado para la publicación de documentos y la interoperabilidad de bases de datos, permite al usuario al que pertenecen los datos en XML, trasladarlos al modelo de datos que desee. Se implementan las sentencias de manipulación de datos que permiten su manejo completo.

- ***La gramática implementada en el SQLmx***. Es importante considerar los aspectos de la gramática SQL estándar, pues con base en ella se especifica la forma en que son representados los datos por el SQLmx**,** ésto es transparente para el usuario, pero importante en la implementación del Módulo XML. Con la gramática se generan las sentencias SQL que modifican las tablas correspondientes, por lo cual debe existir una correcta relación entre los tipos de datos recibidos y la estructura del esquema al que corresponden.

- ***El controlador JDBC del SQLmx.*** Este aspecto es tomado en cuenta, pues se encarga de la interacción con el SQLmx, permitiendo el acceso a la base de

datos a consultar o modificar (las características principales del JDBC de SQLmx, se incluyen en el marco teórico – véase capítulo 2).

- ***La recomendación XML 1.0 REC-xml-20011006***. La investigación realizada está basada en la versión 1.0 de XML, segunda edición de la recomendación del 6 de octubre de 2000, utilizando las reglas necesarias para el funcionamiento del Módulo XML.

- ***DTD internas.*** Las DTD que definen los elementos de las consultas realizadas se generan de forma dinámica y se incluyen dentro del documento XML que se regresa como resultado al cliente, con el fin de evitar su almacenamiento en el servidor, lo que disminuiría el espacio en disco, implicando un mantenimiento de borrado posterior a estos archivos.

- ***Representación DOM como interfaz de programación.*** La API DOM permite estructurar internamente un documento XML en forma de árbol, dando la ventaja de recorrerlo buscando los elementos que se desean obtener, e incluso, agregarle otros nodos según las necesidades.

- ***Internet Explorer 5.0 ó posterior***. Para ejecutar la interfaz de acceso al SQLmx a través de Internet, se recomienda utilizar el navegador Web Internet Explorer 5.0 ó posterior, ya que cuenta con un *parser* XML que permite desplegar en pantalla los documentos que se devuelven como resultado de las consultas realizadas. Netscape 6.0 da soporte a archivos XML, presentándolos con una hoja de estilo que esconde las etiquetas, no obstante se puede emplear cualquier navegador con soporte a XML.

3.5 Resumen

En este capítulo, se ha detallado el análisis y diseño del Módulo XML para acceder al SABD SQLmx a través de Internet, mostrando diagramas UML y diagramas de bloques para ejemplificar la arquitectura construida.

Capítulo 4

Implementación

4.1 Introducción

En este capítulo se explican los procesos principales del desarrollo del Módulo XML, los cuales implican la instalación del entorno de trabajo, la codificación de las clases que conforman a los submódulos de consulta y actualización, así como su implementación usando el lenguaje de programación Java.

4.2 Configuración del entorno de trabajo

La arquitectura del Módulo XML se ha implementado mediante clases Java, agrupadas en un paquete nombrado *moduloxmlsqlmx,* el cual permite al SABD SQLmx recibir sentencias SQL de manipulación de datos a través de Internet. Este conjunto de clases empaquetadas puede también utilizarse con otros SABD, incluyéndose como una biblioteca que extiende las funciones del SABD.

Para su correcto funcionamiento, se requiere que el equipo de cómputo en que se utilice cuente con el ambiente de ejecución Java[2] y configurar algunas variables de entorno como se muestra a continuación:

```
JAVA_HOME=/disco/jdk1.2
CLASSPATH=$JAVA_HOME/jre/lib:$JAVA_HOME/lib:$JAVA_HOME/jr
e/lib/sparc:/jre1.2/lib
```

Cuando un cliente desea utilizar el Módulo XML para acceder al SABD SQLmx, debe realizar una petición al servidor utilizando la página Web respectiva para consultar o modificar datos. El servidor Web, se conecta a través de *servlet* con el SQLmx. En este caso se utiliza TomCat 3.2.2 del proyecto Jakarta [16] y para configurarlo se coloca en la variable de ambiente CLASSPATH el valor TOMCAT_HOME, que hace referencia a la ruta de instalación**:**

[2] El JDK y el JRE para varias plataformas se puede descargar de la página de SUN Microsystems, http://www.java.sun.com

```
TOMCAT_HOME=/disco/jakarta-tomcat-3.2.2
CLASSPATH=$TOMCAT_HOME/webapps/public_html/WEB-
INF/classes:$TOMCAT_HOME/lib/servlet.jar:$TOMCAT_HOME/lib
/parser.jar
```

Las páginas Web se colocan en un directorio particular, destinado para que sea accesible desde Internet, en este caso se instalaron en:

`/jakarta-tomcat-3.2.2/webapps/public_html/paginas/moduloxml`

y los archivos con extensión *.class*, generados al compilar cada *servlet* se almacenan en el directorio *classes*, dentro de *WEB-INF* del contenedor de *servlets*:

`/jakarta-tomcat-3.2.2/webapps/public_html/WEB-INF/classes`

El contenedor se encarga de pasar las peticiones del cliente al *servlet* y este último de devolver la respuesta.

Un *servlet* extiende la clase abstracta `HttpServlet` (`javax.servlet.http.HttpServlet`), la cual se divide en subclases abstractas que deben sobrescribir algunos de los métodos siguientes [16]:

- `doGet`, si el *servlet* soporta peticiones HTTP GET.
- `doPost`, para peticiones HTTP POST.
- `doPut`, para peticiones HTTP PUT.
- `doDelete`, para peticiones HTTP DELETE.
- `init` y `destroy`, para administrar los recursos que utiliza el *servlet* mientras está activo.
- `getServletInfo`, para que el *servlet* proporcione información referente a él mismo.

Los *servlet* implementados en esta aplicación sobrescriben el método *doPost*, de la siguiente forma:

```
public void doPost(HttpServletRequest req,
HttpServletResponse resp) throws ServletException,
IOException
```

Este método permite al cliente enviar datos de longitud variable al servidor Web, y es útil cuando se desea incluir el tipo de contenido y codificación. Finalmente, se coloca la ruta del *parser* XML que analiza los documentos recibidos:

`CLASSPATH=JAVA_HOME/lib/xerces.jar`

El Módulo XML permite conectarse a cualquier SABD relacional, utilizando para ello los parámetros solicitados en las páginas Web que lo acceden.

4.3 Módulo de consulta

Cuando un usuario desea realizar una consulta a una base de datos perteneciente al SQLmx, debe introducir el nombre de la clase JDBC que maneja el controlador del SABD, así como su URL, el nombre de la base de datos, del usuario, su clave de acceso y la sentencia SQL a ejecutar en la página Web respectiva, misma que activa un *servlet* que se encarga de comunicarse con el Módulo XML.

4.3.1 Proceso realizado en el *servlet* para consultar la base de datos

Para poder procesar una sentencia de consulta (instrucción SQL **select**), el *servlet* debe:

- Conectarse con el SABD SQLmx.
- Ejecutar la consulta.
- Invocar al Módulo XML para crear el documento resultado.
- Regresar el documento XML al cliente.

Si se descomponen cada una de las partes del algoritmo anterior, el proceso queda como sigue:

Conectarse con el SABD SQLmx.

```
INICIO
    Definir un controlador.
    Buscar la clase del controlador.
    Si existe realizar una conexión.
    inicio SI
        Si se conectó:  CONTINUAR.
        Si no:
            Emitir Mensaje.
            TERMINA.
    Si no
       Emitir Mensaje.
    fin SI

FIN
```

Ejecutar la consulta.

```
    INICIO
        Crear una sentencia SQL.
```

Ejecutar la consulta y recuperar el resultado.

FIN

Invocar al Módulo XML para crear el documento XML de resultado.

Para poder crear un documento, el paquete *moduloxmlsqlmx* necesita conocer el resultado de la consulta SQL ejecutada, sus *metadatos* y un nombre de raíz del documento que se genera como resultado. El *servlet* encargado de invocar este paquete realiza el siguiente proceso:

INICIO

Definir un nombre de raíz
Con los datos, metadatos, raíz y sentencia crear el documento XML resultado.

FIN

Regresar el documento XML al cliente.

INICIO

Definir un flujo de escritura para regresarlo como respuesta a la página Web con el documento resultado.
Desplegar el documento XML en el visualizador de Internet.

FIN

El usuario decide si debe guardar el documento XML recibido como respuesta, o consultarlo desde el visualizador. El navegador Internet Explorer utiliza una plantilla XSL (*Extensible Stylesheet Language*, Lenguaje de Hojas de Estilo Extensible) que permite organizar el documento XML, expandiendo o contrayendo el contenido de sus etiquetas.

4.3.2 Proceso para crear el documento XML resultante

El documento XML que se despliega en el visualizador del cliente incluye una DTD, que se genera dinámicamente con el resultado de la consulta. Esta DTD define la estructura semántica y sintáctica de los datos que se devuelven como resultado.

Como resultado del análisis del módulo, los tipos de datos son agregados como atributos del primer elemento XML del documento resultado. Esto es útil cuando se desea conocer el tipo de dato original para procesarlo o convertirlo posteriormente. Colocarlo únicamente en el primer registro evita tener un documento cargado con información redundante para cada elemento, debido a que los elementos que lo componen son homogéneos.

La clase que implementa el código de este proceso se llama *DocumentX* y consta de dos métodos y un constructor. Para ahorrar espacio en disco se incluye la definición de la DTD en el documento; sin embargo, la clase que la implementa permite

generarla interna o externamente; por lo cual, solo es cuestión de pasar los parámetros respectivos al constructor según se requiera.

Utilizando el resultado de la consulta, se crea el documento XML mediante los siguientes pasos:

INICIO

Conocer la raíz del documento y los metadatos para crear la DTD.
Conocer la cantidad de columnas resultado.
Construir un árbol DOM.

FIN

Construcción del árbol DOM para el documento XML

INICIO

Crear un nodo tipo documento.
Agregar un nodo raíz al documento.
CICLO Si hay registros resultado
Crear un Nodo elemento.
Colocarle al Nodo el atributo numero (contiene el número del elemento).
Para cada columna
Crear Nodo elemento para crear la etiqueta.
Colocarle sus atributos.
Agregarle un nodo hoja con el valor que contendrá la etiqueta.
Fin de Para cada columna
Agregar el nodo de etiqueta como hijo al nodo raíz.
FIN DEL CICLO Si hay registros resultado

FIN

Generar el documento XML en base al árbol DOM

INICIO

Colocar declaración XML.
Generar DTD.
CICLO Para cada elemento del árbol DOM
Colocar etiqueta de inicio.
Colocarle sus atributos si tiene.
Colocar contenido, si es elemento volver al ciclo.
Colocar etiqueta de fin.
FIN DEL CICLO para cada elemento del árbol DOM.

FIN

4.3.3 Creación de la DTD interna

En un documento XML resultado de una consulta SQL, se incluye la DTD de manera interna, por lo cual el proceso de su construcción para el módulo de Consulta es el siguiente:

INICIO
Recibir METADATOS y RAIZ.
Conocer la cantidad de columnas resultado.
Construir un árbol con los METADATOS (similar al documento XML, PERO sin DOM), donde la RAIZ es el nodo superior del árbol.
Retornar la DTD al documento XML.
FIN

Construcción del árbol del documento DTD

INICIO
Crear un nodo principal.
Agregar un elemento Raíz al nodo principal.
Crear un Nodo Elemento REGISTRO.
Definir los ATRIBUTOS del elemento REGISTRO.
Agregar el nodo del elemento REGISTRO como hijo a la raíz del árbol.
CICLO Para cada columna METADATO
Crear Nodo Elemento CAMPO.
Colocar a CAMPO ATRIBUTO(s).
A elemento CAMPO agregarle un nodo hoja (#PCDATA).
Añadir a nodo REGISTRO el hijo CAMPO.
FIN DEL CICLO Para cada columna METADATO
Colocar el documento DTD en el documento XML cuando se invoque la DTD.
FIN

Finalmente, la estructura de la DTD para un documento resultado de una consulta queda como se muestra en la Figura 4.1.

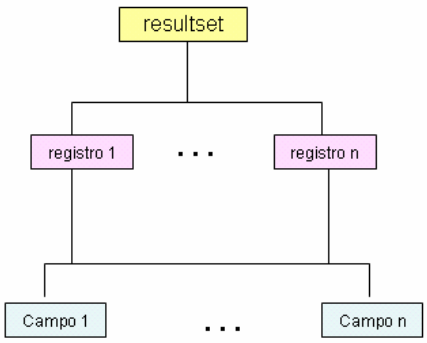

Figura 4.1 Estructura de la DTD interna que se devuelve en el resultado de una consulta.

Un ejemplo de DTD interna generada para la consulta *select * from proveedor* consta de lo siguiente:

```
<!DOCTYPE resultset [                                          1
<!ELEMENT resultset(registro)*>                                2
<!ELEMENT registro(NoProv, NomProv, CiuProv)>                  3
<!ATTLIST registro numero ID #IMPLIED>                         4
<!ELEMENT NoProv (#PCDATA)>                                    5
<!ATTLIST NoProv tipo CDATA #IMPLIED>                          6
<!ELEMENT NomProv (#PCDATA)>                                   7
<!ATTLIST NomProv tipo CDATA #IMPLIED>                         8
<!ELEMENT CiuProv (#PCDATA)>                                   9
<!ATTLIST CiuProv tipo CDATA #IMPLIED>                         10
]>                                                             11
```

La línea número 1 indica, mediante la palabra **DOCTYPE** seguida de *resultset*, que lo que viene a continuación es una DTD. En la línea 2, el modelo de contenido para *resultset* es de cero o más elementos (indicado por el asterisco) llamados *registro.* Un elemento *registro* puede contener a su vez los elementos *NoProv*, *NomProv*, *CiuProv*, los cuales se definen en las líneas 5, 7 y 9 con la palabra **#PCDATA**, que indica que no se subdividirán más. En las líneas 6, 8 y 10, se define el atributo que acompaña a los elementos anteriores, llamado *tipo,* que no es necesario analizar, definiéndose como **CDATA,** la palabra **#IMPLIED** especifica la necesidad de aparecer en el documento como opcional. El elemento *registro* también tiene el atributo *numero*, que por estar definido como de tipo *ID,* indica que no se puede repetir, ya que representa un identificador; la definición de este atributo se observa en la línea 4 de la DTD expuesta. El final de las definiciones de la DTD se marca con los caracteres de la línea 11.

4.4 Módulo de modificación

El módulo de modificación permite realizar cambios en el contenido de una base de datos mediante XML. El usuario, al igual que en las consultas, debe especificar en la página Web: el controlador del SABD, la ubicación de la base de datos, el nombre de la base de datos, el usuario, su clave de acceso, el nombre de la tabla e incluir un archivo XML. En caso que desee agregar registros, el documento XML anexado debe contener los datos que se desea insertar en la tabla; si desea hacer alguna eliminación, se deben incluir los datos que forman la condición para borrar o si prefiere actualizar el contenido de campos de un registro, debe incluir los nuevos datos y las condiciones que seleccionen los registros a modificar. En el siguiente capítulo se muestran algunos ejemplos de la forma de realizar consultas y actualizaciones usando una página HTML conectada al SQLmx.

4.4.1 Proceso realizado en el *servlet* para modificar la base de datos

Para poder procesar una sentencia SQL de manipulación de datos, el *servlet* que responde a la página Web realiza los siguientes pasos:

- Obtener los parámetros introducidos en la página Web. Se validan los parámetros, si son correctos se continua el proceso, de lo contrario se envía el mensaje de error correspondiente.
- Establecer una conexión con el SQLmx.
- Invocar el Módulo XML para realizar el proceso de inserción, borrado o modificación, según lo solicitado.
- Ejecutar en el SQLmx cada sentencia SQL generada por el Módulo XML**.**
- Enviar un mensaje de resultado como página Web hacia el usuario**.**

Si descomponemos cada una de las partes del algoritmo anterior, el proceso queda como sigue:

Obtener los parámetros introducidos en la página Web

La página HTML envía a través del método *doPost* los datos al *servlet* como un solo elemento, por lo que es necesario indicar que la información que proviene de la página Web contiene diferentes tipos de datos; para esto se utiliza la definición:

```
ENCTYPE="multipart/form-data"
```

que especifica que la página Web envía uno o más parámetros y que pueden ser de diferente contenido, que en el caso de la página del Módulo XML son textos y un archivo. Cada parámetro tiene un nombre, el cual es único dentro de una página HTML, para poder delimitar cada uno de ellos se agrega un encabezado que indica su contenido (*content-disposition*) [17] y un parámetro adicional (*name*) con el nombre del elemento dentro de la página HTML. Para un archivo, se agrega su tipo de contenido (*Content-type*), el cual se infiere de la extensión del archivo (véase Figura 4.2), por lo que debe tener extensión *.xml*.

Cada elemento se envía en el orden en que aparece en la página Web, como una parte de un flujo (*flujo multiparte*).

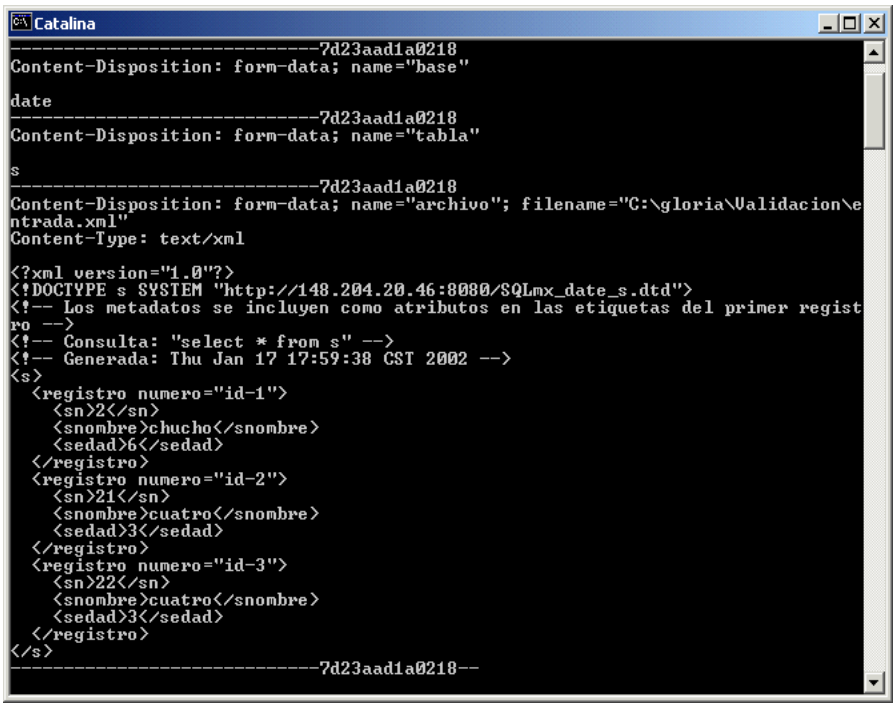

Figura 4.2 Flujo multiparte recibido por el *servlet* de Modificación de Datos.

Al recibir la información es necesario separarla en sus correspondientes elementos, pues el archivo XML contiene información de cabecera (véase Figura 4.2), la cual no va etiquetada conforme a la recomendación XML y que si se pasa al *parser* provoca un resultado de mala formación. La separación se hace usando clases de la *API Struts* [16].

```
MultipartIterator multiparte = new MultipartIterator(req,
    4096, 4096000, "/disco/struts/");
MultipartElement multiElem;
```

La clase `MultipartElement` contiene un método `getFile()`, que permite extraer el archivo, manejando uno temporal en el cual se verifica el tipo de contenido y quita todos los encabezados adicionados al momento del envío. El archivo sin encabezados queda listo para continuar el proceso del *servlet*.

Validación de parámetros

En este paso se verifica que se hayan recibido todos los parámetros necesarios para realizar el proceso, por lo que solo es necesaria una condición que lo afirme:

```
Si no ha existido un error por falta de parámetro
entonces CONTINUAR.
```

Conectarse con el SABD SQLmx (véase sección 4.3.1.- Conectarse con el SABD SQLmx)

Invocar al Módulo XML para realizar el proceso correspondiente. Para realizar el proceso correspondiente, el Módulo XML debe generar la DTD con la que valida el documento XML recibido (véase sección 4.4.3), y posteriormente realizar el proceso de inserción, eliminación o modificación de registros (véase sección 4.4.4).

Ejecutar cada sentencia SQL generada por el Módulo XML en el SQLmx. Esto se realiza mediante la tecnología de conexión a la base de datos JDBC (utilizada en esta aplicación), a través del método que ejecuta sentencias SQL *execute()*, para cada una de las sentencias generadas en el paso anterior.

INICIO
CICLO Para cada sentencia
Ejecutar la sentencia SQL.
FIN DEL CICLO Para cada sentencia
FIN

Enviar mensaje de resultado a la página Web. El *servlet* lleva un contador de las sentencias ejecutadas con éxito, y en caso contrario, retorna un mensaje de error al visualizador para aquellas que no se realizaron.

4.4.2 Consultar el Diccionario de Datos

Una vez hecha la conexión con el SABD, se procede a consultar el diccionario de datos para extraer los *metadatos* de la tabla en la que se insertarán, eliminarán o actualizarán los registros. De acuerdo al tipo de dato, se define la sentencia SQL deseada.

En esta parte, se detallan los algoritmos necesarios para adicionar el Módulo XML al SABD SQLmx, o a otro SABD relacional.

Para SQLmx

INICIO
Buscar en Catálogo de Tablas la tabla.
Si existe
Obtener su identificador.
Si no
Emitir Mensaje.
TERMINAR
Seleccionar del catálogo de columnas aquellas que pertenecen a la tabla mediante su identificador.
CICLO Para cada columna
Se guarda el nombre de la columna y su tipo de dato.
FIN CICLO Para cada columna
FIN

Para otro SABD Relacional

INICIO
Obtener los metadatos de la conexión.
Obtener los metadatos de la tabla que nos interesa.
CICLO Para cada columna
Obtener nombre // columna 4 según especificación JDBC
Obtener tipo / columna 6 según especificación JDBC para getMetaData*/*
FIN CICLO Para cada columna
FIN

4.4.3 Generar la DTD que valide el documento XML recibido

Como XML hace diferencia entre mayúsculas y minúsculas, se debe ser muy cuidadoso en los siguientes aspectos de los parámetros a enviar a la página Web: todo lugar donde aparezca el nombre de la tabla y de la base de datos, deben escribirse exactamente igual, ya que las etiquetas se extraen de la DTD correspondiente a la tabla para que sean correctas**,** de lo contrario no se puede generar el árbol, ya que se verifica que las etiquetas coincidan con el nombre de los campos de la tabla (en caso de que el usuario no conozca el esquema de la tabla, se recomienda realizar primero una consulta al servidor y descargar una DTD que permita conocer el esquema actualizado de la misma). El nombre de la raíz del documento debe ser el nombre de la tabla a la que se agregan, borran o actualizan datos. Por ejemplo, el esqueleto de un archivo XML que pretende insertar campos en una tabla llamada *alumno* de la base de datos *CIC* es el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1”?>                          1
<!DOCTYPE alumno SYSTEM                                              2
"http://148.204.20.46:8080/SQLmx_CIC_alumno.dtd">
<alumno>                                                             3
  <insert>                                                           4
     <sn>2</sn>                                                      5
     <snombre>chucho</snombre>                                       6
     <sedad>6</sedad>                                                7
  </insert>                                                          8
.
.
.
</alumno>                                                            9
```

La primera línea se refiere a la declaración XML, la cual no debe faltar, ya que con ella se identifica el tratamiento que se debe dar al archivo.

En cuanto a la segunda línea, esta corresponde a la declaración DOCTYPE, en donde se pide que el usuario escriba:

```
<!DOCTYPE alumno SYSTEM "http://148.204.20.46:8080/SQLmx_CIC_alumno.dtd">
```

El nombre físico de la DTD se compone agregando la palabra *SQLmx* más el nombre de la base de datos (*CIC*, en el ejemplo*)* y el nombre de la tabla (*alumno*, en el ejemplo), separados por un guión bajo (_) y con extensión *.dtd.* Es necesario tener presente que el nombre asignado en esta declaración y lo especificado en la página de modificación se escriba tal cual, ya que es validado por el servidor Web.

La tercera y novena líneas representan el contenido del documento, con la etiqueta que lo abre y cierra respectivamente; ésta debe ser idéntica a la palabra que sigue a la palabra DOCTYPE de la segunda línea, pues en ella se define la raíz a usar en el documento XML; finalmente, también debe coincidir con el nombre de la tabla en la declaración DOCTYPE.

La etiqueta mostrada en la línea 4 indica la instrucción a realizar con respecto a la operación a la base de datos, que puede ser de inserción, eliminación o actualización (sentencias *insert, delete* o *update*, respectivamente). El resto de las etiquetas, las cuales forman el cuerpo del documento XML, se toman del nombre de los campos de la tabla, para el caso de inserciones. En la actualización de datos y eliminación de registros se permite agregar etiquetas opcionales, útiles para agregar una condición, de modo que no se afecte a todos los registros de una tabla y que se muestran en la Tabla 4.1.

<**sentencia** *comp*="tipo_comparacion"> </**sentencia**>
 	
Ejemplo: que el número de la *parte* sea mayor a diecinueve:
 	
`<sentencia comp=">">`
`  <parte>19</parte>`
`</sentencia>`
	La etiqueta **sentencia** tiene la función de comparar el campo especificado dentro de las etiquetas que contiene anidadas, con el valor del elemento. El tipo de comparación (>,<, =) se especifica en su atributo *comp*. Puede estar o no contenida dentro de una etiqueta *paréntesis.*

<**parentesis** conj="operador_logico"> </**parentesis**>
 	
Ejemplo: que el color sea oro o plata:
 	
`<parentesis conj="or">`
`  <sentencia comp="=">`
`  <color>oro</color>`
`  </sentencia>`
`  <sentencia comp="=">`
`  <color>plata</color>`
`  </sentencia>`
`</parentesis>`
	La etiqueta *paréntesis* agrupa dos etiquetas **sentencia** para realizar comparaciones unidas por operador lógico *and* u *or*, que se indica en su atributo *conj*.

Tabla 4.1 Etiquetas opcionales para las sentencias *update* y *delete*.

Estas etiquetas siguen las mismas normas que equivalen a utilizar un *where* de SQL. Para actualizar valores se proporcionan las etiquetas de la Tabla 4.2.

<**set**> </**set**>
 	
Ejemplo: poner el color a aqua:
 	
`<set>`
`  <color>aqua</color>`
`</set>`
	Con esta etiqueta se especifica la lista de campos de la tabla (cada una como un elemento) y sus valores nuevos, También puede contener etiquetas del tipo *condición.*

<**condicion** *fn*="operador aritmetico">
</**condicion**>
 	
Ejemplo, multiplicar el peso por dos:
 	
`<set>`
`  <condicion fn="*">`
`  <peso>2</peso>`
`  </condicion>`
`</set>`
	La etiqueta **condición** permite realizar operaciones que actualicen el valor un campo. Su atributo *fn* representa el operador aritmético a usar (+,-,*,/) sobre el campo (tomado como primer operador) en la cantidad del valor del elemento que contiene.

Tabla 4.2 Etiquetas para la sentencia *update*.

La validación de los datos recibidos en el documento XML se apega a una DTD que se genera con los nombres de las columnas y la raíz del documento. Esta DTD es externa al documento XML enviado por el usuario, y se almacena en un subdirectorio predefinido, por lo que este directorio contiene tantas DTD como tablas existan en la base de datos.

Los pasos de creación de la DTD pueden resumirse así:

- Recibir COLUMNAS y RAIZ.
- Conocer la cantidad de columnas de resultado.
- Construir un árbol con las COLUMNAS Y RAIZ.
- Retornar la DTD al *servlet*.

El proceso de creación del árbol para la DTD de modificación a una tabla, es similar al de creación de la DTD para el módulo de consultas, con la diferencia de que para modificar, se agregan etiquetas que definan el proceso a realizar, como *insert*, *delete*

y *update*. Para poder insertar datos, el documento XML anexado por el usuario debe contener como etiquetas los nombres de los campos de la tabla; para la condición *update*, es necesario especificar las columnas a modificar y/o una condición que contenga el cálculo a realizar (+,-,/,*), y si se desea se puede incluir una condición donde se defina sobre qué registros va a operar. Esto último es aplicable de igual forma para el caso de eliminación (*delete)* e incluso para poder utilizar varias sentencias agrupadas dentro de etiquetas *parentesis* (véase Figura 4.3).

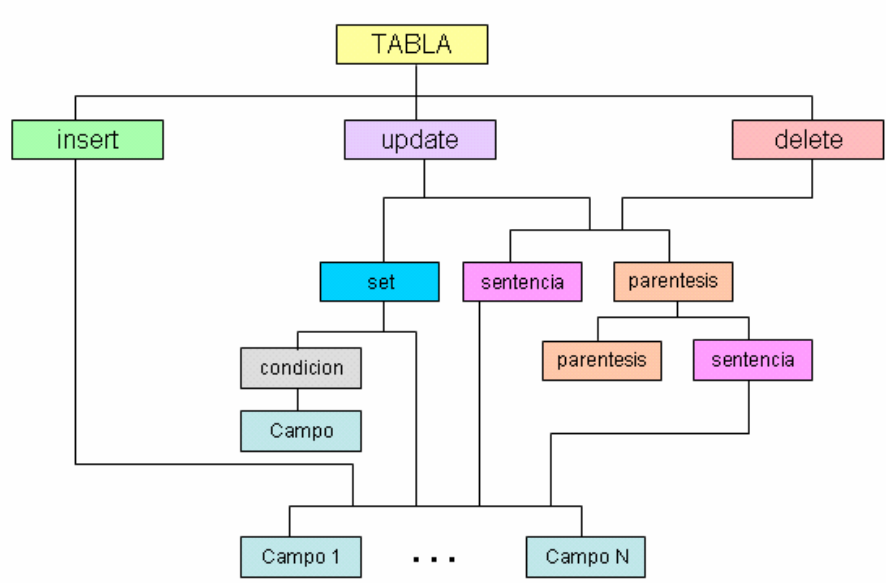

Figura 4.3 Estructura de la DTD para modificar una tabla

4.4.4 Crear el proceso correspondiente para INSERT, DELETE y UPDATE

Para poder crear el proceso correspondiente a la actualización de valores en una tabla, se crea un objeto java (*ProcDocX*) que recibe como parámetro el archivo XML anexado por el usuario y la estructura de los tipos de datos:

INICIO
Validar el Documento XML con la DTD.
Si es válido CONTINUAR.
Si no Emitir Mensaje.
FIN
/ Extraer los datos del árbol DOM. */*
CICLO Para cada elemento
***Inicio Comparar** elemento en caso de que sea igual a:*
INSERT Elaborar sentencia Insert.
DELETE Elaborar sentencia Delete.

UPDATE Elaborar sentencia Update.
Fin de Comparar
Colocar sentencia en conjunto de sentencias.
FIN CICLO Para cada elemento
Regresar conjunto de sentencias SQL.
FIN

4.5 Resumen

En este capítulo, se ha detallado cómo se implementan las funciones que conforman el Módulo XML para acceder al SABD SQLmx a través de Internet. Las clases que implementan los métodos descritos en esta sección, se agrupan en un paquete Java (*moduloxmlsqlmx*) que se configura en el entorno del sistema, trabajando como un componente para el SABD SQLmx. El código fuente se muestra en el anexo D.

Capítulo 5

Pruebas y resultados

5.1 Introducción

En esta fase se busca lograr el correcto funcionamiento de la aplicación desarrollada, por lo cual, se esquematizaron ejemplos de consultas y modificaciones que se realizan a través de las páginas Web diseñadas para tal efecto y que permiten al usuario personalizar el acceso a bases de datos, y utilizar el Módulo XML en la manipulación de datos, para revisar que el trabajo en conjunto con el SABD SQLmx sea el deseado, además de verificar su integración con otro sistema administrador de bases de datos relacional.

5.2 Equipo de cómputo

El equipo utilizado para el desarrollo del proyecto es una PC IBM 300GL, con disco duro de 20GB y 96MB, Sistema Operativo Windows 2000 y JDK 1.3.1.

5.3 Esquema utilizado como ejemplo

El módulo de consulta se prueba utilizando, como ejemplo, la base de datos formada por tres tablas: *proveedor*, *parte* y *provparte*.

Las tablas *proveedor* y *parte*, se identifican de manera única por medio del número de proveedor (*NoProv*) y número de parte (*NoParte*), respectivamente. El significado de un registro de la tabla *provparte* (envío) es que un proveedor suministra la parte especificada en la cantidad registrada con el atributo *cant.*

5.3.1 Diagrama entidad-relación del ejemplo

En la Figura 5.1, se observa que un proveedor puede suministrar muchas partes (tabla *provparte*) en determinada cantidad; la definición de las partes se encuentra en el catálogo de partes.

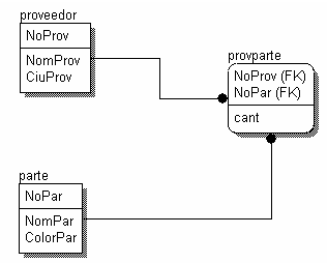

Figura 5.1 Diagrama entidad-relación del ejemplo utilizado para las pruebas.

5.4 Pruebas y resultados

La forma en que el usuario final tiene acceso al Módulo XML del SQLmx es por medio de páginas Web. El servidor Web puede estar en la misma máquina o distinta al servidor de datos.

5.4.1 Consulta a la Base de datos

Las consultas SQL que se muestran a continuación, permiten observar la manera en que el Módulo XML trabaja de forma conjunta con el SABD SQLmx para generar un archivo XML con los registros del resultado. Para ello, se piden en la pantalla de consultas, los datos necesarios para realizar la conexión con la base de datos (véase Figura 5.2).

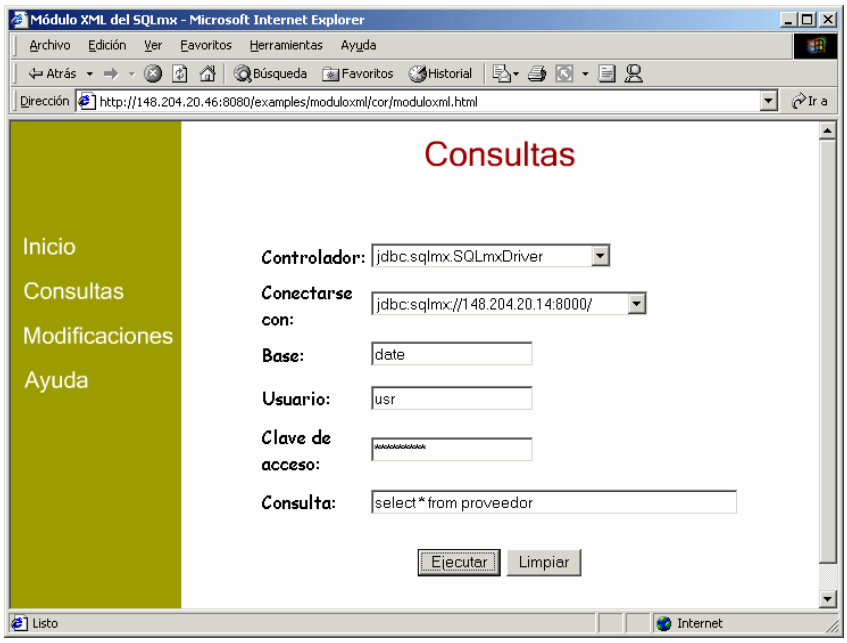

Figura 5.2 Pantalla de consultas.

Consulta: *Select * from tabla*

La Figura 5.3 muestra el resultado de la consulta SQL *select * from proveedor*, observada en el navegador Internet Explorer 5.0. El elemento raíz recibe el nombre *resultset* y puede contener varios o ningún registro. En este caso se observan varios registros (que comienzan con la etiqueta *registro*), todos conteniendo como atributo un número identificador. Los *metadatos* se incluyen únicamente en el primer registro, debido a que cada elemento contenido en el documento XML representa una instancia de la tabla, por lo que los elementos del documento XML comparten los mismos atributos. Incluirlos en cada elemento hace que el documento se vea cargado de información redundante, es por esta razón que sólo aparecen en la etiqueta del primer registro, por lo tanto, se aconseja al usuario revisar el primer elemento para conocer los tipos de datos.

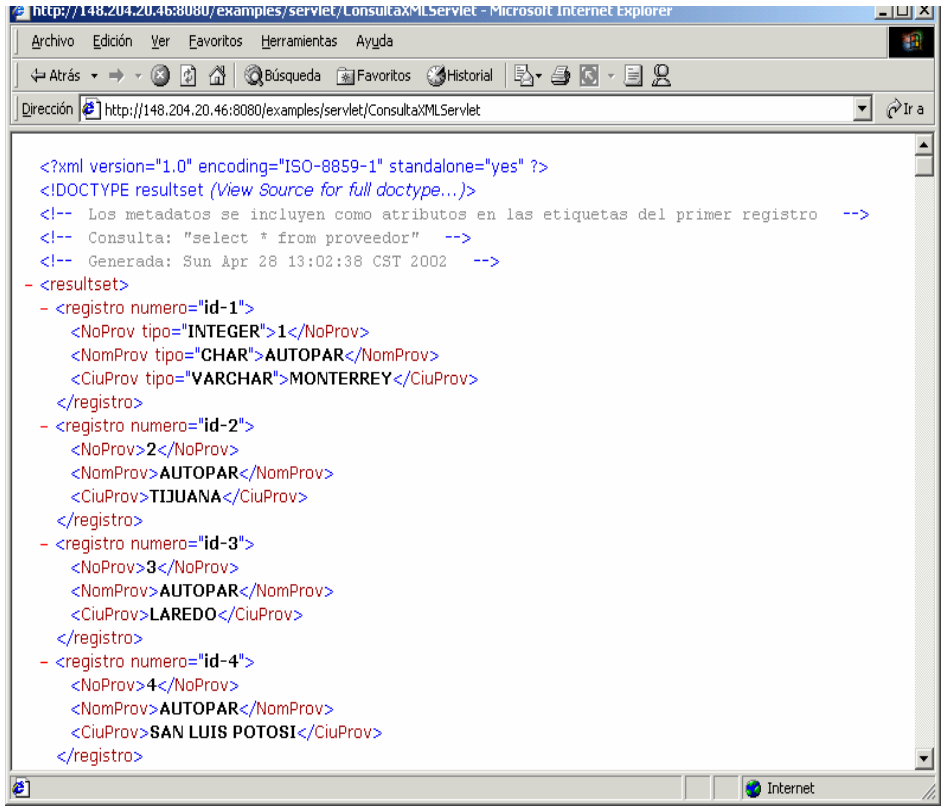

Figura 5.3 Consulta *select * from proveedor.*

La DTD queda dentro del documento XML devuelto y puede apreciarse al ver el código fuente de la página Web (véase Figura 5.4).

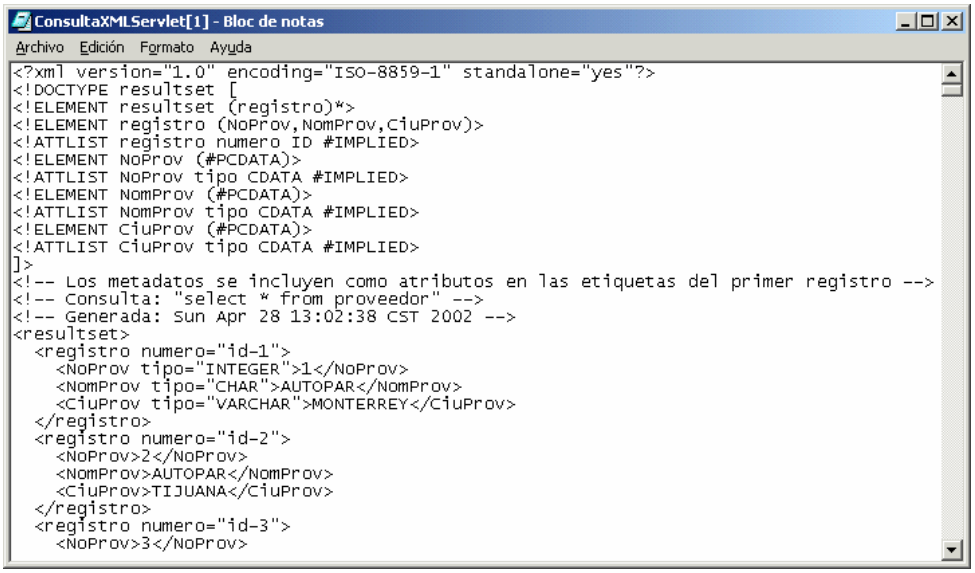

Figura 5.4 DTD interna del Documento XML resultado de la consulta *select * from proveedor*

Consulta: Select *campo1, campo 2, ...* from *tabla* where *condición*

Otra prueba realizada es la obtención de un archivo XML, a partir de datos en una tabla utilizando una condición, como en el ejemplo *select NomProv,CiuProv from proveedor where NoProv<3* (véase Figura 5.5).

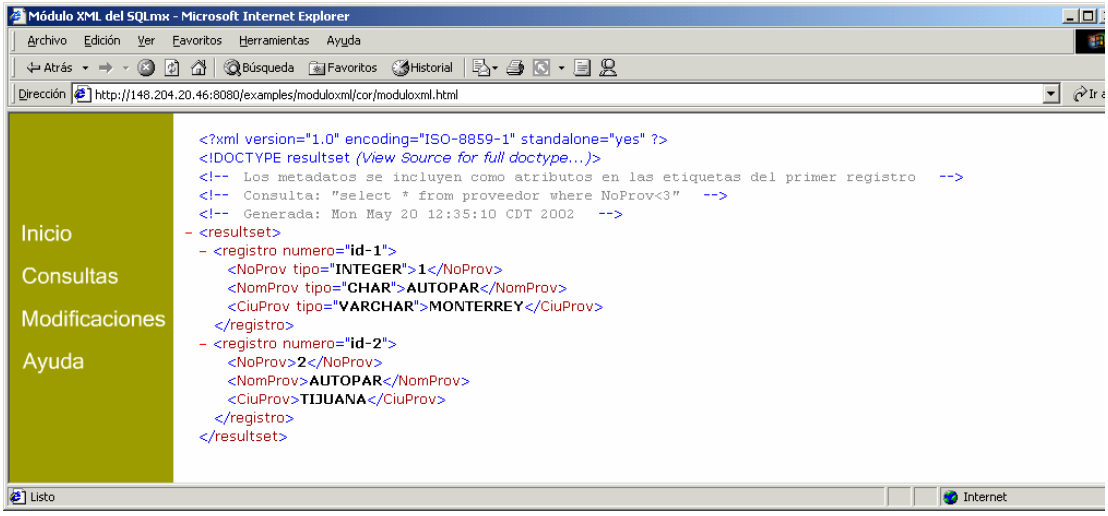

Figura 5.5 Consulta *select NomProv,CiuProv from proveedor where NoProv<3.*

Consulta con resultados nulos

Cuando el resultado no devuelve registros, el documento XML muestra la etiqueta de raíz con una diagonal, que significa que el contenido es vacío (véase Figura 5.6). Este es un resultado válido, ya que si por ejemplo se considera la sentencia SQL: *select NomProv,Ciuprov from Proveedor where NoProv=800*, se tiene que en la tabla proveedor no existe un registro cuyo campo *NoProv* sea igual a 800.

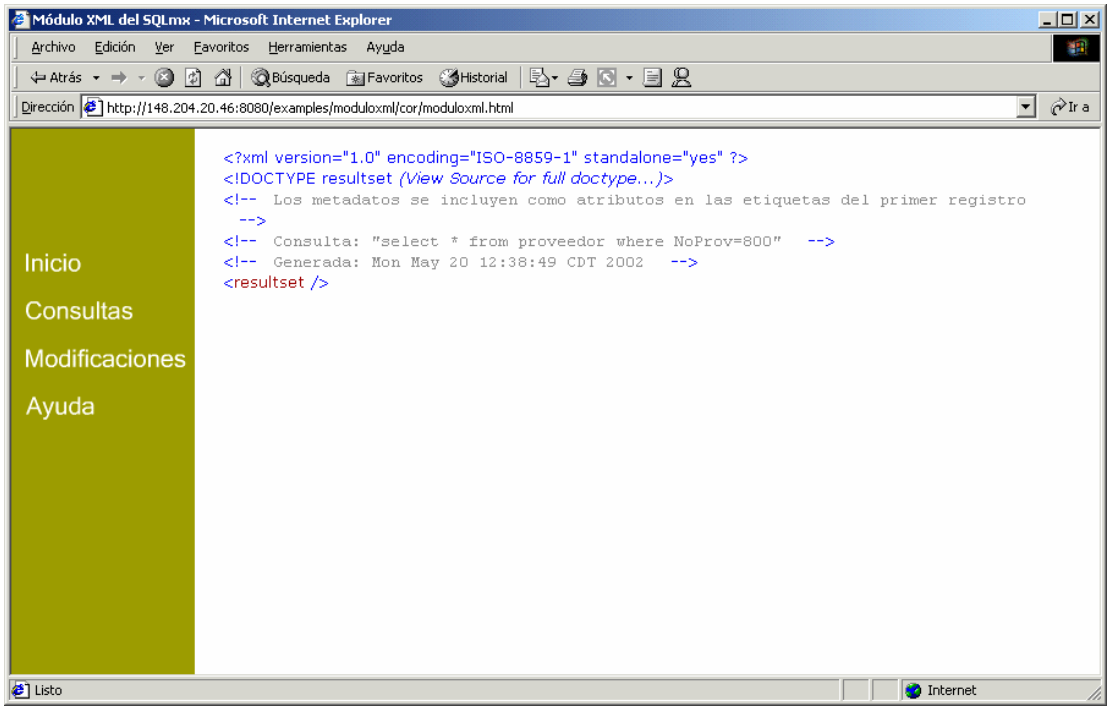

Figura 5.6 Consulta vacía.

5.4.2 Inserción de registros

Otra de las posibilidades que se tiene con el Módulo XML, es la inserción de registros a una tabla por medio de un archivo XML. El usuario sólo tiene que especificar los datos que se le piden en la página Web: el nombre del controlador JDBC, URL del servidor base de datos, nombre de la base de datos, nombre del usuario, su clave de acceso, tabla a modificar y el archivo XML que contiene los datos que van a ser insertados (véase Figura. 5.7).

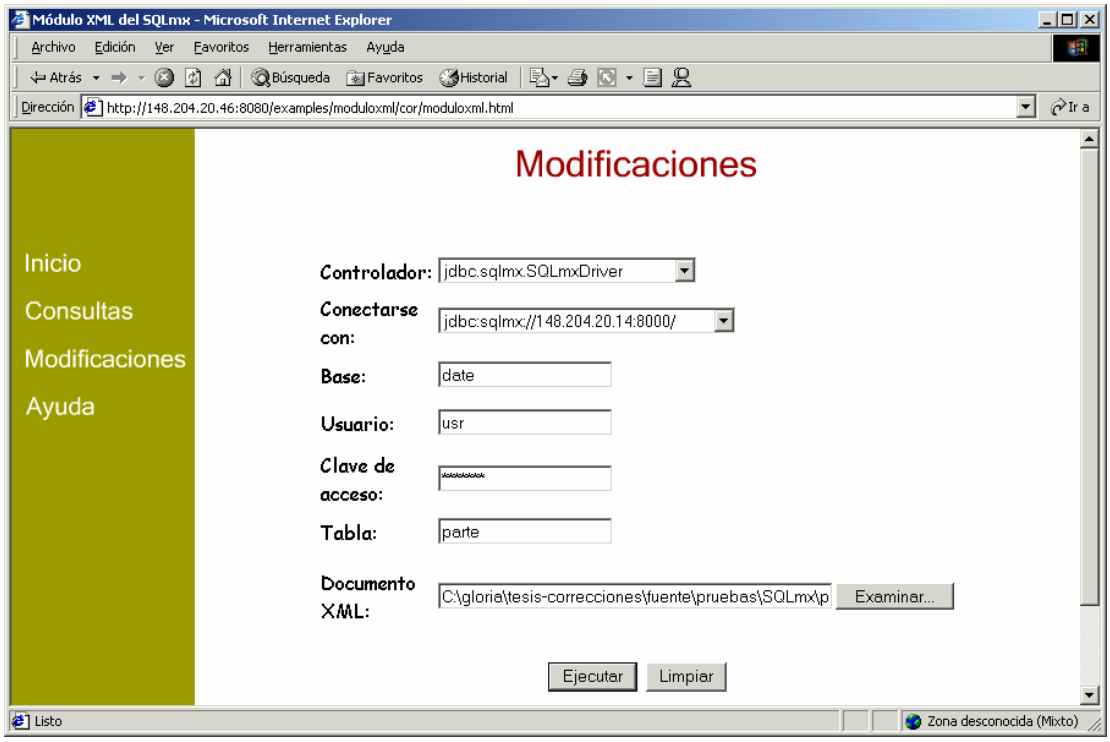

Figura 5.7 Página Web para enviar parámetros y modificar una tabla de la base de datos.

A continuación, se dan algunos ejemplos de pruebas del módulo de **Modificaciones,** para insertar registros, borrarlos o modificar valores de campos.

Inserción de registros a una tabla

El siguiente archivo XML contiene cuatro registros que se desean insertar en la tabla *parte* de la base de datos *date*. El segundo de ellos no incluye el campo *ColorPar* (véase el resultado en la Figura 5.8).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE parte SYSTEM "http://148.204.20.46:8080/SQLmx_date_parte.dtd">
<parte>
  <insert>
    <NoPar>200</NoPar>
    <NomPar>TORNILLO2</NomPar>
    <ColorPar>AMARILLO</ColorPar>
  </insert>
  <insert>
    <NoPar>201</NoPar>
    <NomPar>TUERCA2</NomPar>                /* Falta el elemento ColorPar */
 </insert>
 <insert>
    <NoPar>202</NoPar>
    <NomPar>LLANTA</NomPar>
    <ColorPar>AMARILLO</ColorPar>
  </insert>
 <insert>
    <NoPar>203</NoPar>
    <NomPar>MOTOR</NomPar>
```

```
    <ColorPar>AMARILLO</ColorPar>
  </insert>
</parte>
```

El archivo XML contenía la información de 4 registros y el resultado (véase Figura 5.8), indica que se pudieron insertar todos.

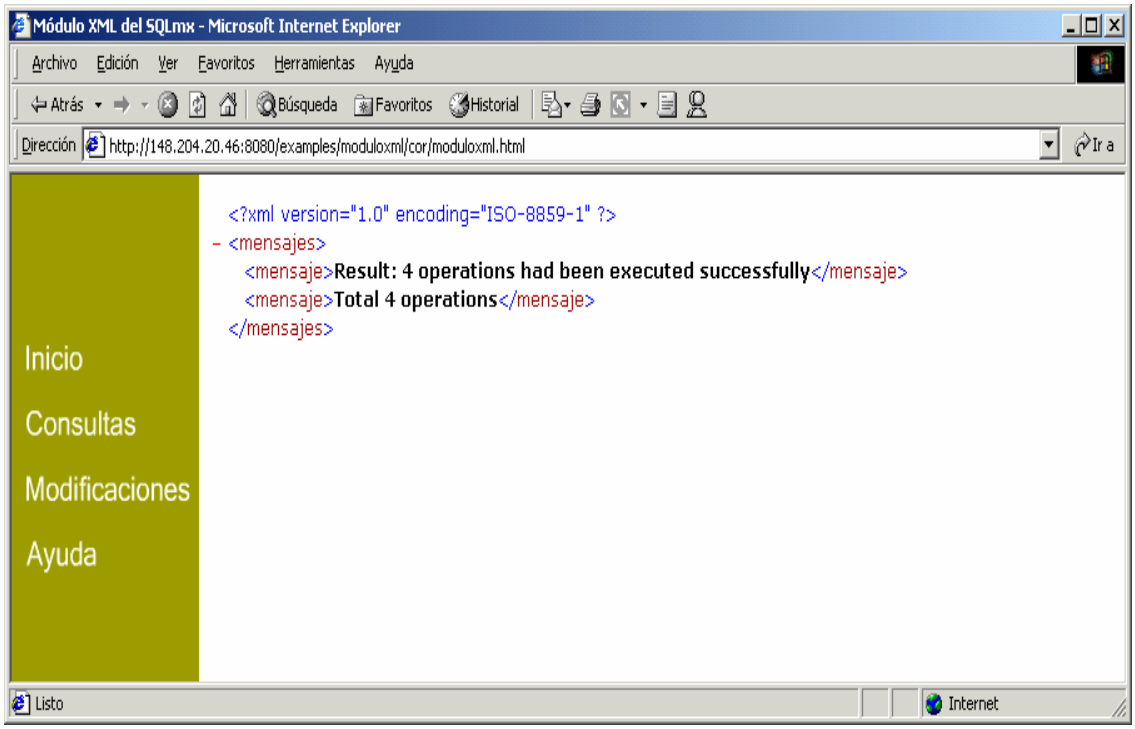

Figura 5.8 Resultado de la operación, después de insertar registros

Ejecutando la sentencia: *select * from parte where NoPar>199*, se puede verificar que la tabla fue actualizada (véase Figura 5.9), en donde la parte 201 no tiene valor en el campo *ColorPar*.

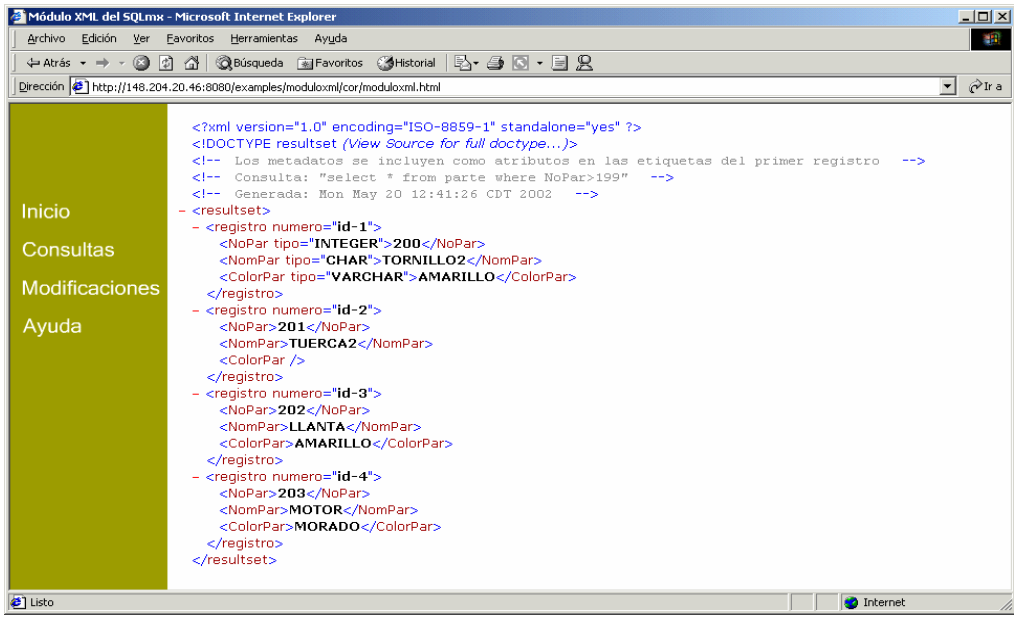

Figura 5.9 Resultado de *select * from parte where NoPar>199* después de la inserción.

Inserción de un documento XML mal formado

Puede darse el caso de que un documento XML esté mal formado o que no sea válido. Por esta razón, el analizador notifica cuando existe un error y envía un mensaje al usuario. Para probar esto, se inserta el siguiente archivo en el que un elemento es diferente en la etiqueta que abre a la que cierra.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE parte SYSTEM "http://148.204.20.46:8080/SQLmx_date_parte.dtd">
<parte>
  <insert>
    <NoPar>200</NoPar>
    <NomPar>TORNILLO2</NomPar>
    <ColorPar>AMARILLO</ColorPar>
  </insert>
  <insert>
    <NoPar>201</NoPar>
    <NomPar>TUERCA2</NomPar>
 </insert>
 <insert>
    <NoPar>202</NoPar>
    <NomPar>LLANTA</Nom>                    /* Etiquetas incongruentes  */
    <ColorPar>AMARILLO</ColorPar>
  </insert>
 <insert>
    <NoPar>203</NoPar>
    <NomPar>MOTOR</NomPar>
    <ColorPar>AMARILLO</ColorPar>
  </insert>
</parte>
```

El módulo XML devuelve un mensaje (véase Figura 5.10), en el que se menciona que el documento está mal formado y que no se puede construir el árbol DOM, y por consiguiente no se pueden generar las sentencias para insertar los registros.

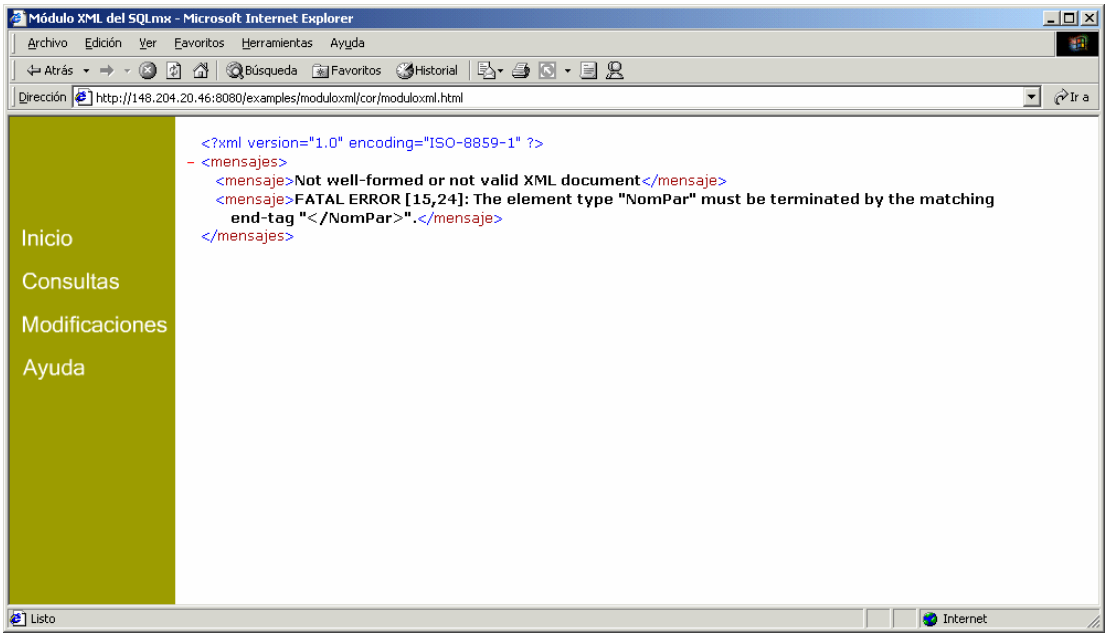

Figura 5.10 Resultado al tratar de introducir un documento XML mal formado.

5.4.3 Eliminación de registros

La eliminación de registros de una tabla usando como condición un archivo XML es otra opción que permite el Módulo XML. El usuario sólo tiene que especificar el archivo XML que contiene los datos que forman la condición WHERE.

Eliminación de registros a una tabla

El siguiente archivo XML contiene las etiquetas *NoPar* y *NomPar* para borrar dos de los registros que se insertaron anteriormente, identificándolos con un número 2 que se agregó en el nombre de la parte.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE parte SYSTEM "http://148.204.20.46:8080/SQLmx_date_parte.dtd">
<parte>
  <delete>
    <parentesis conj="and">
      <sentencia comp=">">
        <NoPar>199</NoPar>
      </sentencia>
      <parentesis conj="or">
        <sentencia comp="=">
          <NomPar>TORNILLO2</NomPar>
```

```
        </sentencia>
        <sentencia comp="=">
          <NomPar>TUERCA2</NomPar>
        </sentencia>
      </parentesis>
    </parentesis>
  </delete>
</parte>
```

Con las indicaciones del documento XML anterior, se desean las partes que se identifiquen con un número mayor a 199 y que se llamen *TORNILLO2* ó *TUERCA2*, por consiguiente, de los cuatro que se insertaron anteriormente, sólo dos cumplen la condición:

DELETE FROM parte WHERE NoPar>199 and (NomPar='TORNILLO2' or NomPar='TUERCA2')

El resultado se muestra en la Figura 5.11, donde aparece un mensaje que indica la cantidad de operaciones exitosas que se han realizado.

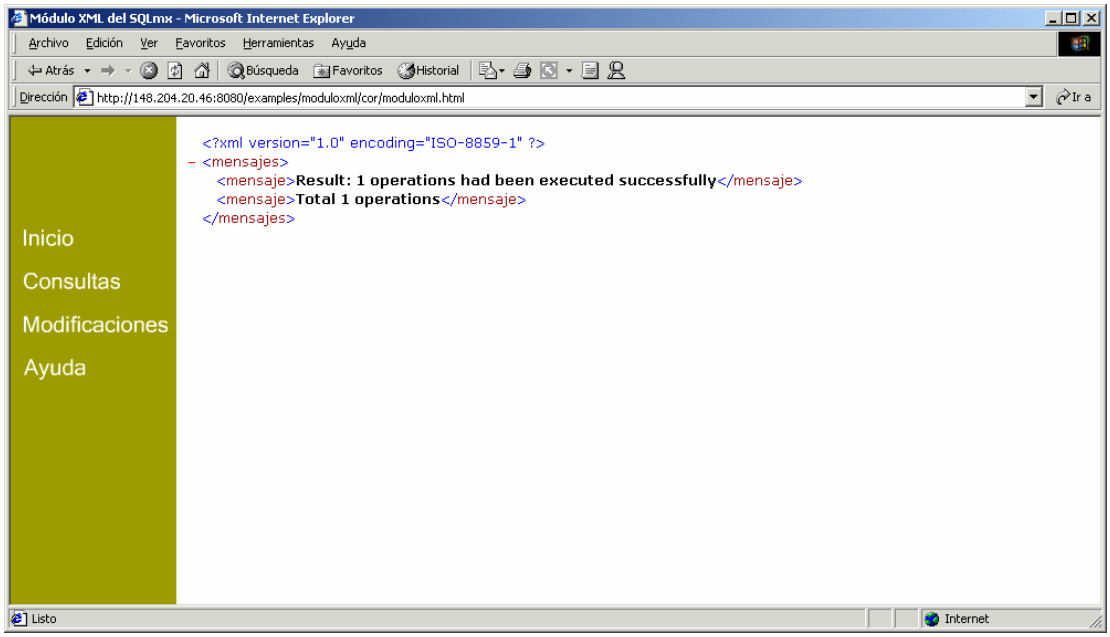

Figura 5.11 Resultado de eliminar registros.

Para verificar la eliminación, se realiza la consulta: *select * from parte where (NomPar='TUERCA2' or NomPar='TORNILLO2');* que retorna el resultado que se muestra en la Figura 5.12.

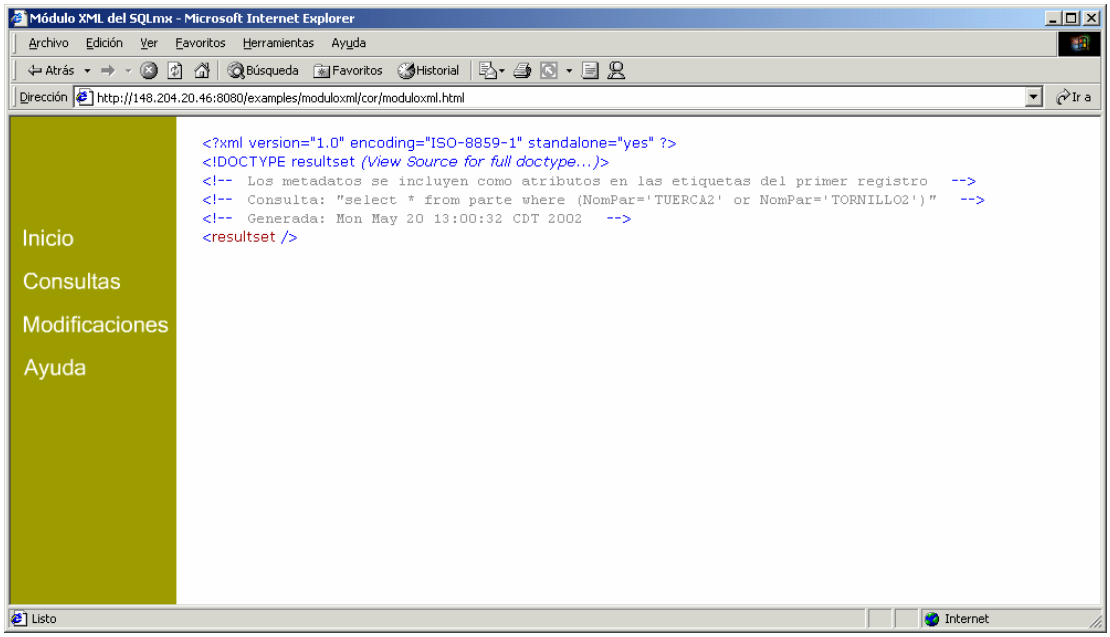

Figura 5.12 Resultado de la consulta *select * from parte where NoPar='TUERCA2' or NomPar='TORNILLO2'* después de actualizarse.

La DTD externa que se genera del lado del servidor para insertar, borrar o actualizar queda estructurada de la siguiente manera:

```
<!ELEMENT parte (update|delete|insert)*>
<!ELEMENT update (set,(sentencia|parentesis)?)>
<!ELEMENT delete (sentencia|parentesis)?>
<!ELEMENT parentesis ((sentencia|parentesis),(sentencia|parentesis))>
<!ATTLIST parentesis conj CDATA #REQUIRED>
<!ELEMENT insert (NoPar?,NomPar?,ColorPar?)>
<!ELEMENT set (condicion|NoPar|NomPar|ColorPar)+>
<!ELEMENT condicion (NoPar|NomPar|ColorPar)>
<!ATTLIST condicion fn CDATA #REQUIRED>
<!ELEMENT sentencia (NoPar|NomPar|ColorPar)>
<!ATTLIST sentencia comp CDATA #REQUIRED>
<!ELEMENT NoPar (#PCDATA)>
<!ELEMENT NomPar (#PCDATA)>
<!ELEMENT ColorPar (#PCDATA)>
```

5.5 Integración con otros SABD Relacionales

En el SABD Access de Microsoft se realizan pruebas de inserción de registros para probar la integridad y consultas en el uso de varias tablas.

Para estas pruebas se han agregado campos a las tablas quedando el diagrama entidad relación como se ve en la Figura 5.13.

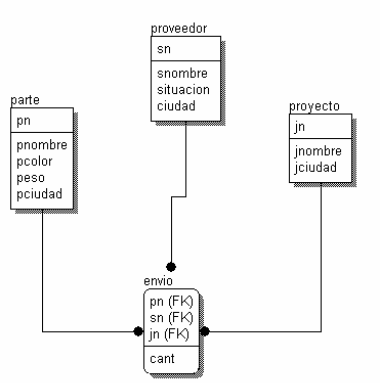

Figura 5.13 Diagrama entidad-relación de la base de datos *date* en el SABD Access.

5.5.1 Insertando datos

Se introduce un documento XML con dos registros, uno de ellos ya existente en la tabla. La Figura 5.14 muestra el resultado en el visualizador. El registro existente genera una excepción, que indica que no se puede insertar el registro, pues ya existe.

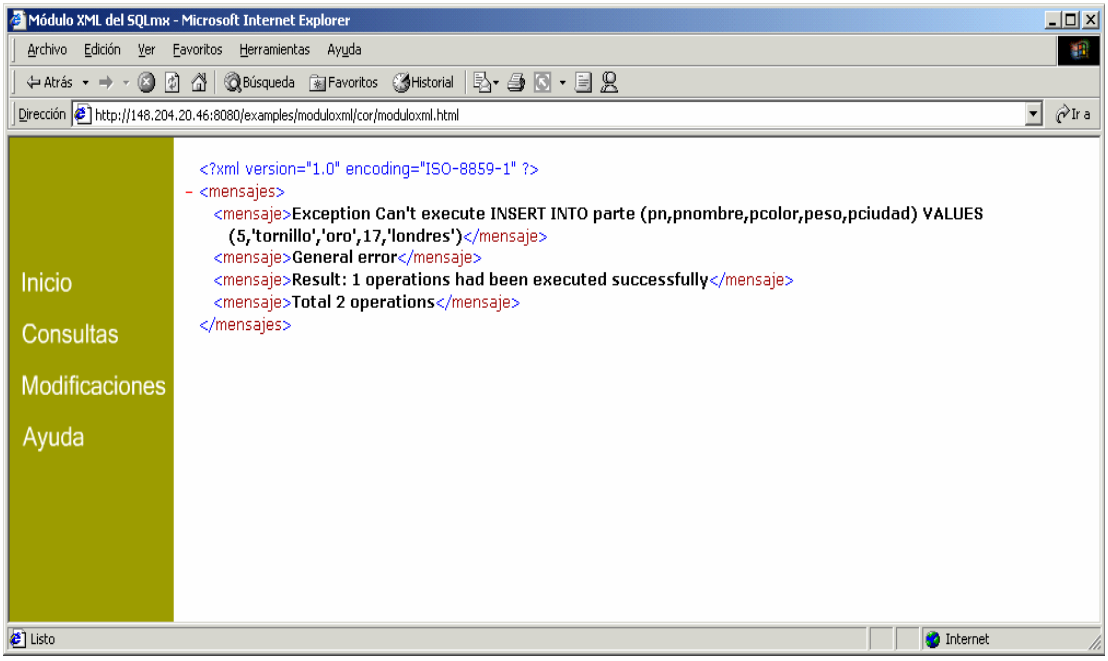

Figura 5.14 Excepción que indica que no se puede insertar el registro, pues ya existe.

Consulta: Select *campo1, campo2, ...* from *tabla1,tabla2, ...* where *condición*

Se ejecuta una prueba con reunión de tres tablas indicando en los resultados, por medio del atributo tipo, el tipo de datos. En la Figura 5.15, se aprecia la consulta y su resultado.

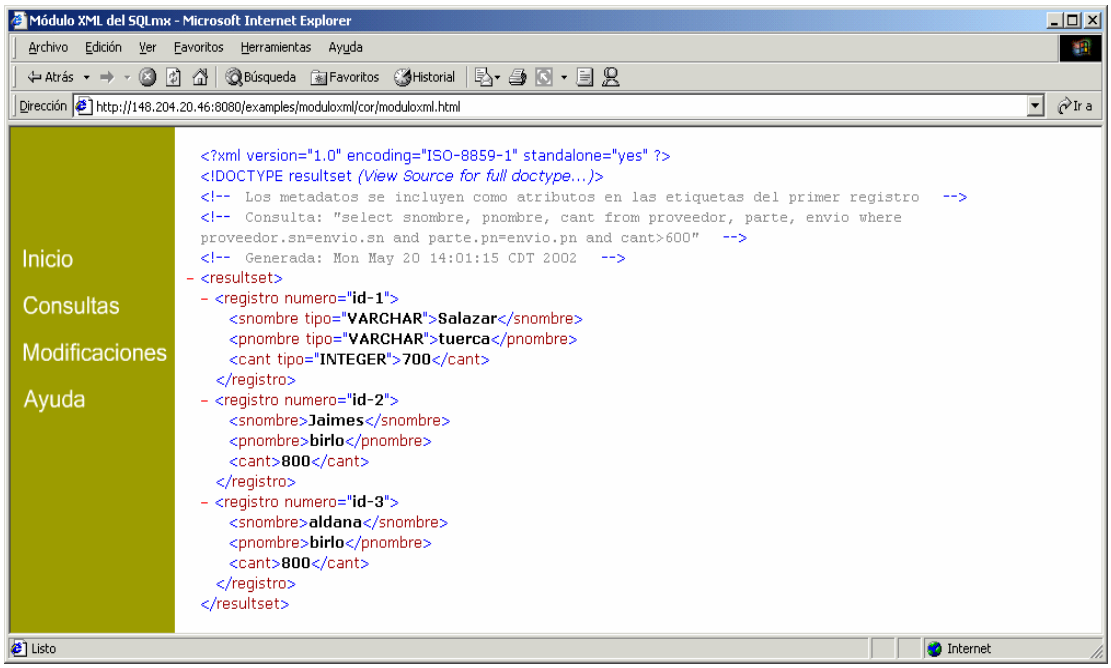

Figura 5.15 Reunión de tres tablas.

5.5.2 Modificación de registros

Para verificar en Access la modificación de datos, primero se insertan en la tabla **parte** los siguientes registros:

Los registros 1 y 2, tienen color oro y plata respectivamente. Se hace la consulta: *select * from parte where pcolor='oro' or pcolor ='plata'*, para comprobar cuantos registros de este tipo existen (véase Figura 5.15).

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE parte SYSTEM "http://148.204.20.46:8080/SQLmx_date_parte.dtd">
<parte>
  <insert>
    <pn>20</pn>
    <pnombre>tornillo</pnombre>
    <pcolor>oro</pcolor>
    <peso>17</peso>
    <pciudad>londres</pciudad>
  </insert>
  <insert>
    <pn>21</pn>
    <pnombre>llanta</pnombre>
    <pcolor>plata</pcolor>
```

```
    <peso>17</peso>
    <pciudad>paris</pciudad>
  </insert>
  <insert>
    <pn>22</pn>
    <pnombre>motor</pnombre>
    <pcolor>morado</pcolor>
    <peso>17</peso>
    <pciudad>venecia</pciudad>
  </insert>
  <insert>
    <pn>23</pn>
    <pnombre>engrane</pnombre>
    <pcolor>morado</pcolor>
    <peso>17</peso>
    <pciudad>atenas</pciudad>
  </insert>
</parte>
```

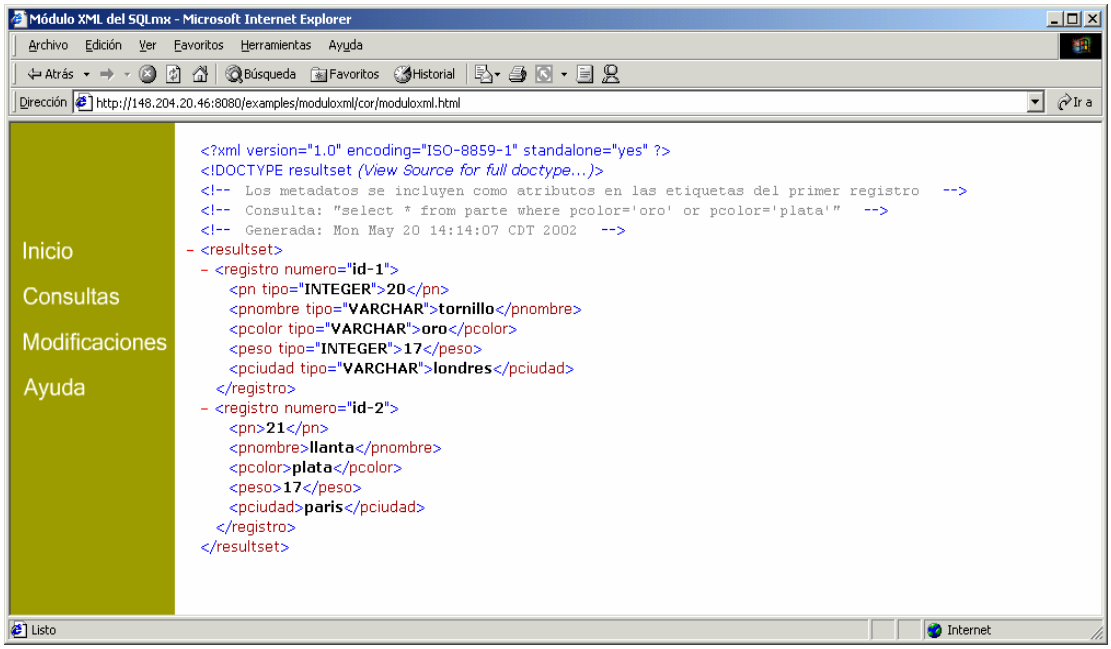

Figura 5.16- Partes de color oro o plata.

El resultado de la consulta que se observa en la Figura 5.16, muestra que existen dos registros que cumplen con la condición de *color ='oro'* o *color ='plata'* y que se identifican por el número de parte (*pn*) 20 y 21, respectivamente.

Ahora, se desean modificar, mediante un archivo XML los registros insertados (*pn* 20, 21, 22, 23), es decir, aquellos cuyo número de parte sea mayor a 19 y que cumplan con la condición de tener color *oro* o *plata*. Además, se desea cambiar el color a *aqua*, la ciudad a *méxico* y duplicar el peso de la pieza.

El archivo XML para realizar la modificación es el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE parte SYSTEM "http://148.204.20.46:8080/SQLmx_date_parte.dtd">
<parte>
  <update>
    <set>
      <pciudad>mexico</pciudad>
      <condicion fn="*">
        <peso>2</peso>
      </condicion>
      <pcolor>aqua</pcolor>
    </set>
    <parentesis conj="and">
      <sentencia comp=">">
        <pn>19</pn>
      </sentencia>
      <parentesis conj="or">
        <sentencia comp="=">
          <pcolor>oro</pcolor>
        </sentencia>
        <sentencia comp="=">
          <pcolor>plata</pcolor>
        </sentencia>
      </parentesis>
    </parentesis>
  </update>
</parte>
```

Los resultados se observan al realizar la consulta: *select * from p where pcolor=’aqua'* (véase Figura 5.17), como puede observarse, para las piezas 20 y 21 que cumplieron con la condición especificada, ambas tenían un peso de 17 y cambió a 34, ciudad a *méxico* y color a *aqua*.

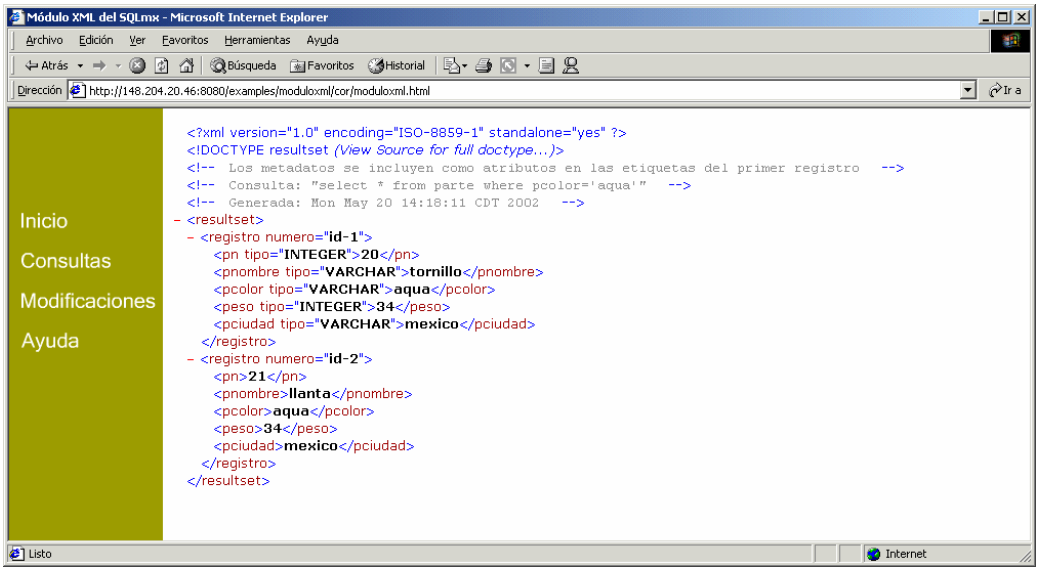

Figura 5.17 Modificación de contenido para la sentencia *update parte set pciudad='mexico', peso=peso*2, pcolor='aqua' where (pn>19 and (pcolor='oro' or pcolor='plata'))* expresada mediante el archivo XML enviado por el cliente.

5.6. Resumen

En este capítulo, se han descrito las pruebas que se han realizado al Módulo XML utilizando el SABD SQLmx y Access de Microsoft. De esta manera, se comprueba su correcto funcionamiento en distintas plataformas y en dos manejadores de bases de datos relacionales, ya que se han obtenido resultados exitosos.

Capítulo 6

Conclusiones

Las ventajas que brinda un SABD es el almacenamiento y la recuperación de datos, teniendo la confianza de que son consistentes, además de contar con rapidez y atomicidad. La evolución de la tecnología informática impone nuevas necesidades a las organizaciones, la importancia de ser competitivo conlleva a utilizar Internet. Al introducir en las empresas e instituciones la facilidad de manejar y publicar datos oportunamente, surge otra necesidad, compartir información y accederla desde sitios remotos, lo cual, demanda no solo buena presentación de la misma sino su reutilización, requiriéndose datos puros. Es aquí donde las compañías distribuidoras de sistemas administradores de bases de datos se han preocupado por agregar a éstos la tecnología XML, la cual viene a satisfacer este requerimiento.

6.1 Logros alcanzados

Se ha implementado la arquitectura del Módulo XML (véase Capítulo 3), alcanzando los siguientes logros:

- El Módulo XML permite utilizar sentencias SQL de manipulación de datos. Para ello, se elabora la DTD respectiva que define semánticamente los elementos que el usuario debe introducir para lograr establecer una comunicación con el SABD.

- Mediante el submódulo de consulta del SQLmx, se logró la recepción de sentencias SQL para consultar bases de datos y generar documentos XML.

- El Módulo XML, genera dinámicamente las DTD internas que corresponden a los documentos XML construidos como resultado de las consultas.

- A partir de un documento XML, el módulo permite :

    - En documentos XML que contienen los datos a insertar, se ejecutan tantas inserciones como registros haya en el documento XML.

  - Borrar o modificar registros afectando a uno o más de la tabla, si los datos de la condición coinciden con más de uno por no tratarse de un campo llave.

  - Se genera una DTD externa que permite validar el contenido de los documentos XML recibidos.

- Se ha desarrollado una interfaz, en la cual el usuario a través de Internet puede interactuar con el SABD SQLmx.

6.2 Aportaciones

- Habilitar al SABD SQLmx para que, mediante la transferencia de datos con formato XML, pueda realizar interoperabilidad con otros SABD.

- Se ha creado una biblioteca de clases que permite realizar la manipulación de datos, para SQLmx y para otros SABD.

- El Módulo XML le permite al SQLmx extender sus capacidades de respuesta a través de Internet, haciéndolo más robusto a las necesidades actuales, de transmisión de datos estandarizados.

- El usuario tiene la posibilidad de especificar la semántica que requiera, estructurando el contenido del resultado de las consultas, de tal manera que se pueda manipular a su conveniencia.

6.3 Trabajos futuros.

- Realizar una aplicación que tome los datos XML y los utilice como entrada a un sistema de información, adaptándolos al modelo de datos de la base de datos que tenga diseñada el sistema de información.

- Habilitar al Módulo XML para que utilice la API SAX para procesar un documento XML, de modo que extienda más la arquitectura del SQLmx.

- Crear un módulo que funcione como repositorio de documentos XML, de tal manera que puedan guardarse como un tipo de dato en SQLmx.

Anexo A

Gramática del SQLmx

En este anexo se muestra la gramática SQL soportada por el SABD SQLmx en la manipulación de datos. Es un subconjunto del SQL/92 y se describe en notación BNF (*Bakus Naur Form*, Forma Normal de Bakus) [3].

A.1 Elementos comunes

- <expresión de tabla> ::= <cláusula from> [<cláusula where>]
- <cláusula from> ::= **FROM** <referencia de tabla> [{, <referencia de tabla>}...]
- <referencia de tabla> ::= <nombre de tabla>
- <nombre de tabla> ::= <identificador de tabla>
- <identificador de tabla> ::= <identificador>
- <identificador> ::= <Idetificador regular>
- <cláusula where> ::= **WHERE** <condición de búsqueda>
- <condición de búsqueda> ::= <término booleano> | <condición de búsqueda> **OR** <término booleano>
- <término booleano> ::= <factor booleano> | <término booleano> **AND** <factor booleano>
- <factor booleano> ::= [**NOT**] <condición de prueba>
- <condición de prueba> ::= <primario booleano> [**IS** [**NOT**] {**TRUE** | **FALSE** | **UNKNOWN**}]
- <primario booleano> ::= <predicado> | (<condición de búsqueda>
- <predicado> ::= <predicado de comparación>

- <predicado de comparación> ::= <constructor de renglón> <operador de comparación> <constructor de renglón>
- <constructor de renglón> ::= <especificación de valor> | {(<especificación de valor> [{,<especificación de valor>}...])}
- <especificación de valor> ::= <expresión de valor>| <constante de cadena de caracteres>
- <expresión de valor> ::= <constante>
- <constante> ::= <constante de cadena de caracteres> | <constante numérica>
- <constante numérica> ::= <constante numérica exacta> | <constante numérica aproximada>
- <constante numérica exacta> ::= [**+**|**-**] {<entero sin signo> [. <entero sin signo>] | <entero sin signo> .| . <entero sin signo>}
- <constante numérica aproximada> ::= <mantisa> E <exponente>
- <mantisa> ::= <constante numérica exacta>
- <exponente> ::= <entero con signo>
- <entero con signo> ::= [**+**|**-**] <entero sin signo>
- <entero sin signo> ::= <dígito>...
- <operador de comparación> ::= **=** | **<** | **>**
- <constante de cadena de caracteres> ::= ' <representación de carácter para cadenas>... '
- <representación de carácter para cadenas> ::= <carácter no apóstrofo ni nueva línea> | <representación de apóstrofo>
- <carácter no apóstrofo ni nueva línea> ::= cualquier carácter diferente del carácter (') y de nueva línea.
- <representación de apóstrofo> ::= ''

La longitud de un identificador no debe ser mayor de 128 caracteres.

- <identificador regular> ::= <letra> [{<subraya> | <letra> | <dígito>}...]
- <dígito> ::=**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**
- <letra> ::= <letra mayúscula> | <letra minúscula>
- <letra mayúscula> ::= **A** | **B** | **C** | **D** | **E** | **F** | **G** | **H** | **I** | **J** | **K** | **L** | **M** | **N** | **O** | **P** | **Q** | **R** | **S** | **T** | **U** | **V** | **W** | **X** | **Y** | **Z**
- <letra minúscula> ::= **a** | **b** | **c** | **d** | **e** | **f** | **g** | **h** | **i** | **j** | **k** | **l** | **m** | **n** | **o** | **p** | **q** | **r** | **s** | **t** | **u** | **v** | **w** | **x** | **y** | **z**
- <subraya> ::= _

A.2 Sentencia select

La sentencia *select* permite recuperar valores de un renglón específico de una tabla. Su sintaxis se da a continuación:

- <especificación de consulta> ::= **SELECT** <lista de select> <expresión de tabla>
- <lista de select> ::= <elemento de select> [{, <elemento de select>}...]
- <elemento de select> ::= {<especificación de valor>

A.3 Sentencia insert

La sentencia *insert* crea o introduce un renglón nuevo a una tabla existente:

- <sentencia insert> ::= **INSERT INTO** <nombre de tabla> [(<lista de columnas de insert>)] {**VALUES** (<lista de valores de insert>)}
- <lista de columnas de insert> ::= <nombre de columna> [{ ,<nombre de columna>}...]
- <nombre de columna> ::= <identificador>

- <lista de valores de insert> ::= <valor de insert> [{,<valor de insert>}...]
- <valor de insert> ::= <constructor de renglón> | **NULL**

Para simplificar, puede decirse que las unidades más pequeñas de una especificación de valor se reducen a: <dígito> <letra> <letra mayúscula>.

A.4 Sentencia delete

La sentencia *delete*, borra renglones de una tabla.

- <sentencia delete> ::= **DELETE FROM** <nombre de tabla> [**WHERE** <condición de búsqueda>]

A.5 Sentencia update

La sentencia actualiza renglones de una tabla.

- <sentencia update> ::= **UPDATE** <nombre de tabla> **SET** <cláusula set> [{, <cláusula set>}...] [**WHERE** <condición de búsqueda>]
- <cláusula set> ::= <columna receptora> = {<especificación de valor> | **NULL**}
- <columna receptora> ::= <nombre de columna>

A.6 Tipos de datos

Los tipos de datos utilizados por SQLmx se definen a continuación:

- <tipo de dato> ::= <tipo cadena de caracteres> | <tipo numérico exacto> | <tipo numérico aproximado>| <tipo fecha-tiempo> | <tipo bit>
- <tipo cadena de caracteres> ::= **CHARACTER** [(<longitud>)] | **CHAR** [(<longitud>)] | **CHARACTER VARYING** [(<longitud>)] | **CHAR** VARYING [(<longitud>)] | **VARCHAR** [(<longitud>)]
- <tipo numérico exacto> ::= **NUMERIC** [(<precisión> , <escala>)]| **DECIMAL** [(<precisión> , <escala>

)]] | **DEC** [(<precisión>, <escala>)] | **MONEY** | **INTEGER** | **SMALLINT** | **LONGINT** | **SERIAL**

- <tipo numérico aproximado> ::= **FLOAT** [(<precisión>)] | **REAL** | **DOUBLE PRECISION**
- <longitud> ::= <entero sin signo>
- <precisión> ::= <entero sin signo>
- <escala> ::= <entero sin signo>
- <tipo fecha-tiempo> ::= **DATE** | **TIME** [(<precisión>)][**WITH TIME ZONE**]
- <tipo bit> ::= **BIT** [(<longitud>)] | **BIT VARYING** [(<longitud>)]
- <entero sin signo> ::= <dígito>...
- <dígito> ::=**0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Son palabras reservadas: AND, BIT, BIT VARYING, CHAR VARYING, CHAR, CHARACTER, CHARACTER VARYING, DATE, DEC, DECIMAL, DELETE FROM, DOUBLE PRECISION, FALSE, FLOAT, FROM, INSERT INTO, INTEGER, LONGINT, MONEY, NOT, NULL, NUMERIC, OR, REAL, SERIAL, SET, SMALLINT, TIME, TRUE, UNKNOWN, UPDATE, VALUES, VARCHAR, WHERE, WITH TIME ZONE.

Anexo B

Comparación entre parser XML

OASIS (*Organization for Advancement of Structured Information Systems*, Organización para el Avance de Sistemas de Información Estructurada) ha elaborado un conjunto de pruebas para medir las capacidades de algunos *parser*, que consisten en tener más de mil documentos válidos y no válidos y verificar si los *parser* los procesan como tal, es decir, aceptar los bien formados y válidos y rechazar el resto.

De las características que esta organización evaluó, se muestran las de interés y utilidad para esta tesis (véase Tabla B1).

Característica	IBM XML4J Apache Xerces	Sun Project X	Microsoft MSXML	Oracle XML *Parser* for Java	James Clark XP
Buena formación	✓	✓	✓	✓	✓
Validación	✓	✓	✓	✓	×
XML-Schema	✓	×	×	×	×
XSL-T	Con Lotus	×	✓	✓	×
Java	✓	✓	×	✓	✓
DOM Nivel 1	✓	✓	✓	✓	×
DOM Nivel 2	✓	×	×	×	×

Tabla B1.- Evaluación de *parser* XML[3].

Las características consisten en evaluar:

Buena formación.- Revisar que el formato de etiquetado de los documentos sea correcto de acuerdo a la recomendación XML.

Validación.- Comprobar que los documentos cumplan con una definición de elementos.

XML-Schema.- Verificar la definición de los elementos que componen los documentos dada en XML.

[3] Los resultados de la tabla se tomaron de http://www.oasis-source.org.

XSL-T.- Si revisa transformaciones de XML a XSL.

Java.- Si el *parser* tiene interfaces para el lenguaje Java.

DOM Nivel 1.- Si el *parser* permite crear y manipular elementos con las interfaces del modelo DOM nivel 1 representándolos en memoria en una estructura de árbol. *DOM*

Nivel 2.- Si el *parser* incluye las interfaces de la API DOM nivel 2 que dan soporte a *nombres de espacio (Namespaces).*

Para desarrollar un proyecto que implique XML, es importante elegir el *parser* adecuado, para el propósito de esta investigación se elige Xerces de Apache, por ser de *código abierto*, *parser validador*, que utiliza interfaces de Java, analizar transformación XSL-T y cumplir con los requerimientos necesarios para dar soporte a la aplicación realizada.

Anexo C

El Modelo de Objeto de Documento

DOM (*Document Object Model*, Modelo de Objeto de Documento) del W3C, es una API para la estructura de un documento que busca facilitar a los programadores el acceso y eliminación de componentes, añadir o editar su contenido, acceder fácilmente a atributos y estilo, y trabajar con todos los navegadores y servidores, sobre todas las plataformas. Los programadores pueden cambiar el lenguaje de programación, pero no necesitan hacer cambios en su modelo de programación.

DOM es una especificación, que permite definir analizadores basados en esta especificación [11].

DOM genera un árbol jerárquico en memoria del documento en XML, considerando un nodo para cada elemento del documento, y permite que a través del *parser* sea manipulada la información.

Las ventajas son las siguientes:

1. Un nodo (información) puede agregarse en cualquier punto del árbol.
2. La información de un nodo puede ser eliminada en cualquier punto del árbol.
3. Los incisos 1 y 2, se ejecutan sin incurrir en las penalidades o limitaciones de manipular un archivo de alguna otra manera.

El árbol jerárquico, se representa con los siguientes elementos:

El objeto Node

El objeto central de DOM es *Node*. Los nodos son objetos genéricos en el árbol y un objeto *Node* define propiedades que sirven para recorrer el árbol:

Propiedad	Definición
nodeType	Código que representa el tipo de objeto.
parentNode	Padre del objeto *Node* actual.
childNode	Lista de hijos para el objeto *Node* actual.

firstChild	Primer hijo del objeto *Node.*
lastChild	Último hijo del objeto *Node.*
previousSibling	Nodo que precede inmediatamente al actual.
nextSibling	Nodo que sigue inmediatamente al actual.
attributes	Lista de atributos del *Node* actual.
nodeName	Nombre del nodo.
nodeValue	Valor del nodo.

El objeto Document

Document es el elemento superior en un árbol, hereda de *Node* y agrega:

- *documentElement*, que es el elemento superior del elemento, y
- *doctype*, que es el tipo de documento.

El objeto Element

Es el descendiente de *Node*. Define la propiedad *tagName*, con base a su nombre de etiqueta y dos métodos específicos:

- *getElementByTagName()* devuelve un objeto de *NodeList*, que es la lista de todos los descendientes del elemento con un nombre de etiqueta dado.
- *normalize()* reorganiza los nodos de texto que están por debajo del elemento, de manera que se encuentren separados solo por etiquetas.

El objeto Text

Devuelve el texto del nodo actual. Asume que el nodo es un elemento mediante la función *getText().*

El objeto Attribute

Contiene un par *nombre-valor* donde *nombre* es el nombre del atributo y *valor* especifica el contenido del nombre. Define los métodos:

- *getValue()*devuelve el nombre del atributo.
- *getName()* devuelve el valor del atributo.

El objeto ProcessingInstruction

Es una instrucción de procesamiento como puede ser la declaración *xml* o una instrucción de procesamiento personalizada.

El objeto Comment

Se pueden definir comentarios dentro del árbol mediante objetos de este tipo.

El objeto CDATASection

Es una objeto que contiene información de una sección del documento que no es analizada por un *parser* .

El objeto DocumentType

En este nodo se declara el tipo de documento. El nodo recibe el nombre del elemento raíz que aparece en la declaración *DOCTYPE.*

El objeto Entity

Representa un nodo que hace referencia a una entidad en una DTD.

El objeto Notation

Representa un nodo que hace referencia a una notación en una DTD.

Anexo D

Código fuente

D.1 *Servlet* para consultar la base de datos

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.sql.*;
import moduloxmlsqlmx.*;

public class ConsultaXMLServlet extends HttpServlet {
    /** valores recibidos de la página web */
    private String controlador=null;
    private String url=null;
    private String base = null;
    private String usuario = null;
    private String pass="";
    private String consulta = null;
    /** atributos para realizar la consulta*/
    private String conBD;
    private Statement  sentencia;
    private Connection con;
    private String mensajeError;

  public void init(ServletConfig config) throws ServletException {
    super.init(config);
  }
  public void doPost(HttpServletRequest req, HttpServletResponse resp)throws
ServletException, IOException {

    mensajeError=new String("");
```

```
    controlador=req.getParameter("controlador").trim();
    if ((controlador==null) || controlador.equals(""))
mensajeError=mensajeError.concat(" Driver missing ");


    url=req.getParameter("url").trim();
    if ((url==null) || url.equals("")) mensajeError=mensajeError.concat("URL
missing");


    base = req.getParameter("base").trim();
    if ((base==null) || base.equals("")) mensajeError=mensajeError.concat("
Data Base name missing ");


    usuario=req.getParameter("usuario").trim();
    if ((usuario==null) || usuario.equals(""))
mensajeError=mensajeError.concat(" User missing ");


    pass=req.getParameter("pass").trim();
    if ((usuario==null)) pass="";


    consulta =  req.getParameter("consulta").trim();
    if ((consulta==null) || consulta.equals(""))
mensajeError=mensajeError.concat(" Statement missing ");


    PrintWriter out = new PrintWriter(resp.getWriter());
    resp.setContentType("text/xml");
    try{
      Class.forName(controlador);
    }catch(ClassNotFoundException e){
      out.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>");
      out.println("<mensajes>");
      out.println("<mensaje>");
      out.println("Exception : Driver "+controlador+ " missing");
      out.println("</mensaje>");
      out.println("</mensajes>");
      out.close();
      return;
    }
    try{
      conBD=url+base;
      con = DriverManager.getConnection(conBD,usuario,pass);
```

```
    }catch(SQLException e){
      out.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>");
      out.println("<mensajes>");
      out.println("<mensaje>");
      if (mensajeError.equals(""))
           out.println("Can't connect to "+ conBD);
      else  out.println(mensajeError);
      out.println("</mensaje>");
      out.println("</mensajes>");
      out.close();
      return;
    }
    try{
      sentencia = con.createStatement();
      ResultSet rs = sentencia.executeQuery(consulta);
      ResultSetMetaData rsmd = rs.getMetaData();
      Raiz raiz=new Raiz("resultset");
      DocumentX ax = new DocumentX(rsmd,rs,raiz,consulta);
      ax.print(out);
      rsmd = null;
      rs = null;
      sentencia.close();
    }catch(SQLException e){
      out.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>");
      out.println("<mensajes>");
      out.println("<mensaje>");
      out.println("Exception"+ e.getMessage());
      out.println("</mensaje>");
      out.println("</mensajes>");
      out.close();
      try{con.close();}catch(SQLException ex){}
      return;
      }
    try{con.close();}catch(SQLException ex){}
    out.close();
  }
}
```

D.2 *Servlet* para modificar la base de datos

```
import javax.servlet.http.*;
import org.xml.sax.*;
import javax.servlet.*;
import java.io.*;
import java.util.*;
import moduloxmlsqlmx.*;
import java.sql.*;
import org.apache.struts.upload.*;

public class ModificaXMLServlet extends HttpServlet{

private String controlador=null;
    private String url=null;
    private String base = null;
    private String usuario = null;
    private String pass="";
    private String conBD;
    private Statement  sentencia;
    private String mensajeError;
    private String tabla = null;
    private String archivo = null;

  public void init(ServletConfig config) throws ServletException {
    super.init(config);
  }

  public void doPost(HttpServletRequest req, HttpServletResponse resp)
              throws ServletException, IOException {

   PrintWriter out = resp.getWriter();

    resp.setContentType("text/xml");
    out.println("<?xml version=\"1.0\" encoding=\"ISO-8859-1\"?>");
    out.println("<mensajes>");
    out.println("<mensaje>");
    MultipartIterator multiparte = new MultipartIterator(req, 4096, 4096000,
"c:\\gloria\\struts\\");
    MultipartElement multiElem;
```

```
    File file = null;


    while((multiElem = multiparte.getNextElement()) != null){
      String cad = multiElem.getName();
      if (cad.equals("controlador")) controlador=
multiElem.getValue().trim();
      if (cad.equals("url")) url= multiElem.getValue().trim();
      if(cad.equals("base"))  base = multiElem.getValue().trim();
      if(cad.equals("usuario"))  usuario = multiElem.getValue().trim();
      if(cad.equals("pass"))  pass = multiElem.getValue().trim();
      if(cad.equals("tabla")) tabla = multiElem.getValue().trim();
      if(cad.equals("archivo")) { if(multiElem.isFile()) file =
multiElem.getFile();}
     }


    mensajeError=new String("");


    if ((controlador==null) || controlador.equals(""))
mensajeError=mensajeError.concat(" Driver missing ");
    if ((url==null) || url.equals("")) mensajeError=mensajeError.concat("URL
missing");
    if ((base==null) || base.equals("")) mensajeError=mensajeError.concat("
Data Base name missing ");
    if ((usuario==null) || usuario.equals(""))
mensajeError=mensajeError.concat(" User missing ");
    if ((pass==null)) pass="";
    if ((tabla==null) || tabla.equals(""))
mensajeError=mensajeError.concat(" Table missing ");
    if ((file==null)) mensajeError=mensajeError.concat(" File missing ");


    if (mensajeError.equals("")) {
      FileReader fr = new FileReader(file);
      Connection conexion=null;
      Statement sentencia;
      ProcDocX ix;
      DocTypesX dtx;
      try{
        Class.forName(controlador);
      }catch(ClassNotFoundException e){
        out.println("Exception: Driver " + controlador + " missing");
        out.println("</mensaje>");
```

```
        out.println("</mensajes>");
        out.close();
        return;
      }

      try{
        conBD=url+base;
        conexion = DriverManager.getConnection(conBD,usuario,pass);
       }catch(SQLException e){
        if (mensajeError.equals(""))
           out.println("Exception: Can't connect to "+ conBD);
        else  out.println(mensajeError);
        out.println("</mensaje>");
        out.println("</mensajes>");
        out.close();
        return;
       }
/** Proceso realizado para el SQLmx */
try{
 LinkedList colnames =  new LinkedList();
 if (controlador.equals("jdbc.sqlmx.SQLmxDriver")){
     //try{
        sentencia = conexion.createStatement();
        boolean ce;
        String obtentabid = new String("select tabid from CatTables where
tabname='" + tabla + "'");
        ResultSet rs = sentencia.executeQuery(obtentabid);
        rs.next();
        String tabid = rs.getString(1).trim();
        sentencia.close();
        try{conexion.close();}catch(SQLException ex){}
        dtx = new DocTypesX();
        try{
          conexion = DriverManager.getConnection(conBD,usuario,pass);
         }catch(SQLException e){
           if (mensajeError.equals(""))
             out.println("Exception: Can't connect to "+ conBD);
           else  out.println(mensajeError);
           out.println("</mensaje>");
```

```
           out.println("</mensajes>");
           out.close();
           return;
          }
        sentencia = conexion.createStatement();
        String obtencols = new String("select colname,datatype from
CatColumns where tabid=" + tabid);
        ResultSet columnas = sentencia.executeQuery(obtencols);
        ResultSetMetaData metadatos = columnas.getMetaData();
        while(columnas.next()){
          String colData = columnas.getString(1).trim();
          String typeData = columnas.getString(2).trim();
          int tipoData = Integer.parseInt(typeData);
          switch(tipoData){
            case 1:
              typeData = "INTEGER";
              break;
            case 2:
              typeData = "SMALLINT";
              break;
            case 3:
              typeData = "LONGINT";
              break;
            case 4:
              typeData = "FLOAT";
              break;
            case 5:
              typeData = "REAL";
              break;
            case 6:
              typeData = "DOUBLE PRECISION";
              break;
            case 7:
              typeData = "NUMERIC";
              break;
            case 8:
              typeData = "DECIMAL";
              break;
            case 9:
```

```
              typeData = "MONEY";
              break;
            case 10:
              typeData = "CHAR";
              break;
            case 11:
              typeData = "VARCHAR";
              break;
            case 12:
              typeData = "DATE";
              break;
            case 13:
              typeData = "TIME";
              break;
            case 14:
              typeData = "SERIAL";
              break;
            case 15:
              typeData = "BIT";
              break;
            case 16:
              typeData = "BIT VARYING";
              break;
            default:
              typeData = "CHAR";
          }
          System.out.println(colData + ":" + typeData);
          dtx.add(colData,typeData);
          colnames.add(colData);
        }
        sentencia.close();
       }   /** Fin del If para SQLmx

/** Para otros SABD  **/
   else {

        DatabaseMetaData dbmd = conexion.getMetaData();
        ResultSet columnas = dbmd.getColumns(null,null,tabla,null);
```

```
        dtx = new DocTypesX();
        int posicion = 0;
        while(columnas.next()){
          posicion = 1;
          String colData = columnas.getString(4).trim();
          String typeData = columnas.getString(6).trim();
          System.out.println(colData + ":" + typeData);
          dtx.add(colData,typeData);
          colnames.add(colData);
        }
        if(posicion == 0){
          out.println(tabla+" not exist ");
          out.println("</mensaje>");
          out.println("</mensajes>");
          out.close();
          try{conexion.close();}catch(SQLException ex){}
          return;
        }
      } /**Fin del else para otro SABD  */



       PrintWriter pwdtd = new PrintWriter(new
FileWriter("c:\\tomcat4.0.1\\webapps\\ROOT\\SQLmx_" + base + "_" + tabla +
".dtd"));
        DocumentD ad = new DocumentD(colnames,new Raiz(tabla));
        ad.print(pwdtd);
        pwdtd.close();
        //columnas=null;


      }catch(SQLException e){
        out.println("Exception: Can't get eschema for " + tabla
+e.getMessage());
        out.println("</mensaje>");
        out.println("</mensajes>");
        out.close();
        try{conexion.close();}catch(SQLException ex){}
        return;
      }
      try{
```

```
      ix = new ProcDocX(fr,dtx);
      fr.close();
      }catch(SAXException e){
       out.println("Not well-formed or not valid XML document");
       out.println("</mensaje>");
        out.println("<mensaje>");
        String cad = e.getMessage();
        out.println(sustituye(cad));
        out.println("</mensaje>");
        out.println("</mensajes>");
        out.close();
        fr.close();
        try{conexion.close();}catch(SQLException ex){}
        return;
      }
      Iterator ite = ix.getIterator();
      int contador = 0;
      int total = 0;
      while(ite.hasNext()){
        total++;
        String cadIns = (String)ite.next();
        System.out.println(cadIns);
        try{
          sentencia = conexion.createStatement();
          sentencia.execute(cadIns);
          sentencia.close();
          contador++;
        }catch(SQLException e){
          out.println("Exception Can't execute " + cadIns);
          out.println("</mensaje>");
          out.println("<mensaje>");
          out.println(e.getMessage());
          out.println("</mensaje>");
          out.println("<mensaje>");
        }
      }
      try{conexion.close();}catch(SQLException ex){ }
      out.println("Result: "+ Integer.toString(contador) + " operations had
been executed successfully "  );
```

```
      out.println("</mensaje>");
      out.println("<mensaje>");
      out.println("Total "+ total + " operations"  );
      out.println("</mensaje>");
      out.println("</mensajes>");
      out.close();

    }//fin del If mensajeError()
   else{
        out.println(mensajeError);
        out.println("</mensaje>");
        out.println("</mensajes>");
        out.close();
   }
  }// fin del doPos

  private String sustituye(String cad){
    int indice = cad.indexOf('<');
    if(indice < 0) return cad;
    String temp = cad.substring(0,indice);
    temp = temp.concat("&lt;");
    temp = temp.concat(cad.substring(indice + 1));
    return temp;
  }
}// fin de la clase
```

D.3 Clases del paquete moduloxmlsqlmx

Clase AttributeD

```
package moduloxmlsqlmx;

/**
  Clase que construye un atributo para un elemento
  de un documento DTD.
*/

public class AttributeD implements Comparable{
  private String nombre;
  private String tipo;
  private String valor;

  /**
    Crea un atributo
  */
  public AttributeD(String nombre, String tipo, String valor){
    // Validar tipo y valor
    this.nombre = nombre;
    this.tipo = tipo;
    this.valor = valor;
  }

  /**
    Obtiene el nombre del atributo
  */
  public String getName(){
    return nombre;
  }

  /**
    Obtiene el tipo del atributo
    Ejemplo: CDATA
  */
  public String getType(){
```

```
    return tipo;
  }

  /**
    Obtiene el valor del atributo
    Ejemplo: #REQUIRED
  */
  public String getValue(){
    return valor;
  }

  /**
    Compara dos atributos para la interfaz Comparable
  */
  public int compareTo(Object obj){
    return nombre.compareTo(((AttributeD)obj).getName());
  }
}
```

Clase DataTypesX

```
package moduloxmlsqlmx;

/**
  Clase que verifica los tipos de una base
*/

class DataTypeX{
  private int tipo;

  /**
    Crea un objeto DataTypeX
  */
  public DataTypeX(String tipoCamp){
  tipoCamp=tipoCamp.toUpperCase();
    if(tipoCamp.equals("CHARACTER") || tipoCamp.equals("CHAR") ||
      tipoCamp.equals("CHARACTER VARYING") || tipoCamp.equals("CHAR
VARYING") ||
      tipoCamp.equals("VARCHAR") || tipoCamp.equals("BPCHAR"))
      tipo = 1;
    else if(tipoCamp.equals("NUMERIC") || tipoCamp.equals("DECIMAL") ||
      tipoCamp.equals("DEC") || tipoCamp.equals("MONEY") ||
      tipoCamp.equals("INTEGER") || tipoCamp.equals("SMALLINT") ||
tipoCamp.equals("INT") || tipoCamp.equals("INT4") ||
      tipoCamp.equals("LONGINT") || tipoCamp.equals("SERIAL") ||
      tipoCamp.equals("FLOAT") || tipoCamp.equals("REAL") ||
      tipoCamp.equals("DOUBLE PRECISION") || tipoCamp.equals("BIT") ||
      tipoCamp.equals("BIT VARYING"))
      tipo = 2;
    else if(tipoCamp.equals("DATE") || tipoCamp.equals("TIME") ||
      tipoCamp.equals("TIME WITH TIME ZONE"))
      tipo = 3;
    else
      tipo = 0;
  }

  /**
    Verifica si es caracter o cadena
```

```
  */
  public boolean isChar(){
    if(tipo == 1)
      return true;
    return false;
  }

  /**
    Verifica si es numerico
  */
  public boolean isNumeric(){
    if(tipo == 2)
      return true;
    return false;
  }

  /**
    Verifica si es fecha o tiempo
  */
  public boolean isDate(){
    if(tipo == 3)
      return true;
    return false;
  }
}
```

Clase Declaration

```
package moduloxmlsqlmx;

/**
  Esta clase construye la declaracion XML
  que identifica a un documento como de este tipo
  Regla No. 23 de la recomendación.
*/

public class Declaration{
  private String delimitaInicio=new String("<?xml");
  private String delimitaFin=new String("?>");
  private String signo=new String("=");
  private String version;
  private String codificacion;
  private boolean independiente;
  private String indepen;

  /**
    Crea una declaracion que especifica la version de XML "1.0".
  */
  public Declaration(){
    version=new String("1.0");
  }

  /**
    Crea una declaracion que especifica la version de XML usada.
  */
  public Declaration(String ver){
    version=ver;
  }

  /**
    Crea una declaracion que especifica la version de XML usada
    y la codificación codif.
  */
  public Declaration(String ver, String codif){
```

```
    version=ver;
    codificacion=codif;
  }

  /**
    Crea una declaracion que especifica la version de XML usada
    y si es STANDALONE. Regla 23, 24 y 32.
  */
  public Declaration(String ver, boolean indep){
    version=ver;
    independiente=indep;
    if(independiente) indepen=new String("yes");
    else indepen= new String("no");
  }

  /**
    Crea una declaracion que especifica la version de XML usada,
    la codificación codif y si es STANDALONE.
  */
  public Declaration(String ver, String codif, boolean indep){
    version=ver;
    codificacion=codif;
    independiente=indep;
    if(independiente) indepen=new String("yes");
    else indepen= new String("no");
  }

  /**
    Devuelve una cadena con el prolog.
  */
  public String getDec(){
    String declara;
    declara=new String(delimitaInicio+" version"+signo+"\""+version+"\"");
    if(codificacion!=null){
      declara=declara.concat("
").concat("encoding").concat(signo).concat("\"").concat(codificacion).concat
("\"");
    }
    if(indepen!=null){
```

```
      declara=declara.concat("
").concat("standalone").concat(signo).concat("\"").concat(indepen).concat("\
"");
    }
    declara=declara.concat(delimitaFin);
    return declara;
  }
}
```

Clase DocTypesX

```
package moduloxmlsqlmx;
import java.util.*;

/**
  Clase para guardar los tipos de las etiquetas de un documento XML.
*/

public class DocTypesX{
  private TreeMap tipos;

  /**
    Crea un objeto DocTypesX.
  */
  public DocTypesX(){
    tipos = new TreeMap();
  }

  /**
    Agrega una etiqueta
  */
  public void add(String etiq, String tipo){
    tipos.put(etiq,tipo);
  }

  /**
    Retorna el tipo de una etiqueta
  */
  public String getType(String etiq){
    return (String)tipos.get(etiq);
  }
}
```

Clase DomParserX

```
package moduloxmlsqlmx;
import org.w3c.dom.*;
import java.io.*;
import org.xml.sax.*;
import org.apache.xerces.parsers.DOMParser;

/**
  Manejador de Excepciones para el parser
*/

public class DomParserX implements ErrorHandler{

  private DOMParser parser = new DOMParser();

  /**
    Construye una Manejador de Excepciones para validar un documento
  */
  public DomParserX() throws SAXException{
    parser.setFeature( "http://xml.org/sax/features/validation", true);
    parser.setErrorHandler(this);
  }

  /**
    Analiza el documento nombreDocXML
  */
  public void parse(String nombreDocXML) throws SAXException, IOException{
    parser.parse(nombreDocXML);
  }

  /**
    Analiza el documento de fuente
  */
  public void parse(InputSource fuente) throws SAXException, IOException{
    parser.parse(fuente);
  }
```

```
/**
  Configura el analizador con la caracteristica
*/
public void configure(String caracteristica, boolean valor)
throws SAXNotRecognizedException, SAXNotSupportedException {
  parser.setFeature( caracteristica, valor );
}

/**
  Devuelve un objeto Document de DOM despues del analisis
*/
public Document getDocument(){
  return parser.getDocument();
}

/**
  Indica si el analizador incluye o no espacios en blanco
*/
public void setIncludeIgnorableWhitespace(boolean val)
  throws SAXNotRecognizedException, SAXNotSupportedException{
  parser.setIncludeIgnorableWhitespace(val);
}

/**
  Maneja los warnings del analizador
*/
public void warning(SAXParseException ex) throws SAXException{
  String men = new String("WARNING [" + ex.getLineNumber() + ","
            + ex.getColumnNumber() + "]: " + ex.getMessage());
  throw new SAXException(men);
}

/**
  Maneja los errores del analizador
*/
public void error(SAXParseException ex) throws SAXException{
  String men = new String("ERROR [" + ex.getLineNumber() + ","
            + ex.getColumnNumber() + "]: " + ex.getMessage());
```

```
    throw new SAXException(men);
  }

  /**
    Maneja los errores fatales del analizador
  */
  public void fatalError(SAXParseException ex) throws SAXException {
    String men = new String("FATAL ERROR [" + ex.getLineNumber() + ","
              + ex.getColumnNumber() + "]: " + ex.getMessage());
    throw new SAXException(men);
  }
}
```

Clase DocumentD

```
package moduloxmlsqlmx;
import java.io.*;
import java.sql.*;
import java.util.*;

/**
  Clase que crea un documento DTD
*/

public class DocumentD{
  private Raiz raiz;
  private ElementD princ;
  private int tipodtd;

  /**
    Crea un documento DTD para select
  */
  public DocumentD(ResultSetMetaData metaDatos, Raiz raiz){
    this.raiz = raiz;
    tipodtd = 1; //select
    buildTree(metaDatos);
  }

  /**
    Crea un documento DTD para insert, update y delete
  */
  public DocumentD(LinkedList rs, Raiz raiz){
    this.raiz = raiz;
    tipodtd = 2; // insert,delete,update
    buildTree(rs);
  }

  private void buildTree(ResultSetMetaData metaDatos){
    try{
      int canColumnas = metaDatos.getColumnCount();
      princ = new ElementD(raiz.getName());
```

```
    ElementD registro = new ElementD("registro");
    registro.addAttrib(new AttributeD("numero","ID","#IMPLIED"));
    princ.addChild(registro);
    ElementD campo;
    for(int j = 1 ; j <= canColumnas ; j++){
        campo = new ElementD(metaDatos.getColumnName(j).trim());
        //LISTA: atributos
        campo.addAttrib(new AttributeD("tipo","CDATA","#IMPLIED"));
        //FIN LISTA
        campo.addChild(new ElementD("#PCDATA"));
        registro.addChild(campo);
    }
  }catch(SQLException e){System.err.println(e);}
}

private void buildTree(LinkedList rs){
  princ = new ElementD("tabla");
  ElementD insert = new ElementD("insert");
  ElementD updateset = new ElementD("set");
  ElementD sentencia = new ElementD("sentencia");
  princ.addChild(insert);
  princ.addChild(updateset);
  princ.addChild(sentencia);
  ElementD condicion = new ElementD("condicion");
  updateset.addChild(condicion);
  sentencia.addAttrib(new AttributeD("comp","CDATA","#REQUIRED"));
  condicion.addAttrib(new AttributeD("fn","CDATA","#REQUIRED"));
  ElementD campoI;
  ElementD campoU;
  ElementD campoS;
  ElementD campoC;
  Iterator ite = rs.iterator();
  while(ite.hasNext()){
    String nuevoCamp = (String)ite.next();
    campoI = new ElementD(nuevoCamp);
    campoU = new ElementD(nuevoCamp);
    campoS = new ElementD(nuevoCamp);
    campoC = new ElementD(nuevoCamp);
```

```
      campoS.addChild(new ElementD("#PCDATA"));
      insert.addChild(campoI);
      updateset.addChild(campoU);
      sentencia.addChild(campoS);
      condicion.addChild(campoC);
    }
  }
  /**
    Imprime un documento DTD en el archivo nomArch.
  */
  public void print(PrintWriter pw){
    if(tipodtd == 1)
      printD(pw,princ);
    else
      printD2(pw,princ);
  }
  private void printD(PrintWriter pw, ElementD elem){
    Iterator ite;
    ElementD elemTemp;
    AttributeD atrib;
    ite = elem.getChildren();
    if(ite.hasNext()){
      pw.print("<!ELEMENT " + elem.getName() + " (");
      while(ite.hasNext()){
        elemTemp = (ElementD)ite.next();
        pw.print(elemTemp.getName());
        if(elem.getName().equals("insert"))
          pw.print("?");
        if(ite.hasNext()){
          if(elem.getName().equals("set")                                    ||
elem.getName().equals("sentencia") ||
             elem.getName().equals("condicion")) pw.print("|");
          else pw.print(",");
        }
      }
      pw.print(")");
      if(elem.getName().equals(raiz.getName()))
        pw.print("*");
      if(elem.getName().equals("set"))
```

```
        pw.print("+");
    }
    else return;
    pw.println(">");
    ite = elem.getAttributes();
    if(ite.hasNext()){
      pw.print("<!ATTLIST " + elem.getName() + " ");
      while(ite.hasNext()){
        atrib = (AttributeD)ite.next();
        pw.print(atrib.getName()  +  "  "  +  atrib.getType()  +  "  "  +
atrib.getValue());
        if(ite.hasNext()) pw.print(" ");
      }
      pw.println(">");
    }
    ite = elem.getChildren();
    while(ite.hasNext()){
      elemTemp = (ElementD)ite.next();
      printD(pw,elemTemp);
    }
  }
  private void printD2(PrintWriter pw, ElementD elem){
    Iterator ite;
    ElementD elemTemp;
    pw.println("<!ELEMENT " + raiz.getName() + " (update|delete|insert)*>");
    pw.println("<!ELEMENT update (set,(sentencia|parentesis)?)>");
    pw.println("<!ELEMENT delete (sentencia|parentesis)?>");
    pw.println("<!ELEMENT                                              parentesis
((sentencia|parentesis),(sentencia|parentesis))>");
    pw.println("<!ATTLIST parentesis conj CDATA #REQUIRED>");
    ite = elem.getChildren();
    while(ite.hasNext()){
      elemTemp = (ElementD)ite.next();
      printD(pw,elemTemp);
    }
  }
}
```

Clase DocumentX

```
package moduloxmlsqlmx;
import java.io.*;
import java.util.Date;
import org.w3c.dom.*;
import java.sql.*;
import org.apache.xerces.dom.*;
import org.apache.xerces.domx.*;

/**
  Objeto que genera un documento XML
*/

public class DocumentX{

  private Document doc;
  private Raiz raiz;
  private String consulta;
  private DocumentD docDtd;

  /**
    Crea un documento XML a partir del conjunto de resultados
    generado por una consulta.
  */
  public DocumentX(ResultSetMetaData metaDatos, ResultSet datos, Raiz raiz,
String consulta){
    this.raiz = raiz;
    this.consulta = consulta;
    docDtd = new DocumentD(metaDatos,raiz);
    buildTree(metaDatos,datos);
  }

  private void buildTree(ResultSetMetaData metaDatos, ResultSet datos){
    try{
      int canColumnas = metaDatos.getColumnCount();
      int renglon = 1;
      doc = new DocumentImpl();
      Element root = doc.createElement(raiz.getName());
      doc.appendChild(root);
```

```
      Element registro;
      Element campo;
      Element texto;
      while(datos.next()){
        registro = doc.createElement("registro");
        registro.setAttribute("numero","id-" + Integer.toString(renglon));
        for(int j = 1 ; j <= canColumnas ; j++){
          campo = doc.createElement(metaDatos.getColumnName(j).trim());
          //LISTA: atributos
          if(renglon == 1)

campo.setAttribute("tipo",metaDatos.getColumnTypeName(j).trim());
          //FIN LISTA
          campo.appendChild(doc.createTextNode(datos.getString(j)));
          registro.appendChild(campo);
        }
        renglon++;
        root.appendChild(registro);
      }
    }catch(SQLException e){System.err.println(e);}
  }
  /**
    Imprime el documento XML en un archivo
  */
  public void print(PrintWriter pw){
    Declaration declaracion = new Declaration("1.0","ISO-8859-1",true);
    DocTypeDec referencia = new DocTypeDec(raiz);
    pw.println(declaracion.getDec());
    pw.println(referencia.getDtdOpen());
    docDtd.print(pw);
    pw.println(referencia.getDtdClose());
    pw.println("<!-- Los metadatos se incluyen como atributos en las
etiquetas del primer registro -->");
    pw.println("<!-- Consulta: \"" + consulta + "\" -->");
    pw.print("<!-- Generada: " + new Date() + " -->");
    XGrammarWriter xgw = new XGrammarWriter(pw);
    xgw.printElement(doc.getDocumentElement());
  }
}
```

Clase DocTypeDec

```
package moduloxmlsqlmx;
/**
  Clase que construye la referencia DOCTYPE para incluir el DTD
  Ejemplo: <!DOCTYPE resultset SYSTEM "resultset.dtd">
  Regla No. 28.
*/
public class DocTypeDec{
  private Raiz elementoRaiz;

  /**
    Crea una declaracion DOCTYPE
  */
  public DocTypeDec(){
    elementoRaiz = new Raiz();
  }

  /**
    Crea una declaracion DOCTYPE con raiz
  */
  public DocTypeDec(Raiz raiz){
    elementoRaiz=raiz;
  }

  /**
    Retorna una cadena con la declaracion DOCTYPE para DTD interna.
  */
  public String getDtdOpen(){
    return new String("<!DOCTYPE " + elementoRaiz.getName() + " [");
  }
  /**
   Retorna una cadena con el final de la declaracion DOCTYPE para una DTD
interna
  */
  public String getDtdClose(){
    return new String("]>");
  }
}
```

Clase ElementD

```
package moduloxmlsqlmx;
import java.util.*;

/**
  Clase para crear elementos de un documento DTD.
*/

public class ElementD{
  private String nombre;
  private TreeSet atributos;
  private LinkedList hijos;

  /**
    Crea un elemento para el documento DTD.
  */
  public ElementD(String nombre){
    this.nombre = nombre;
    atributos = new TreeSet();
    hijos = new LinkedList();
  }

  /**
    Obtiene el nombre del elemento.
  */
  public String getName(){
    return nombre;
  }

  /**
    Agrega un hijo al elemento
  */
  public void addChild(ElementD hijo){
    hijos.add(hijo);
  }

  /**
```

```
  Retorna los hijos de un elemento.
*/
public Iterator getChildren(){
  return hijos.iterator();
}

/**
  Agrega un atributo al elemento
*/
public void addAttrib(AttributeD atrib){
  atributos.add(atrib);
}

/**
  Retorna los atributos del elemento
*/
public Iterator getAttributes(){
  return atributos.iterator();
}
```

Clase ProcDocX

```
package moduloxmlsqlmx;

import java.io.*;
import java.util.*;
import org.w3c.dom.*;
import org.apache.xerces.parsers.*;
import org.xml.sax.*;
import org.apache.xerces.dom.*;

/**
  Procesa un documento XML para generar sentencias SQL
*/
public class ProcDocX{

  private Document doc;
  private Iterator ite;
  private DocTypesX tipos;

  /**
    Construye un objeto ProcDocX
  */
  public ProcDocX(Reader flujo,DocTypesX tipos) throws SAXException{
      try{
        this.tipos = tipos;
        DomParserX domp = new DomParserX();
        domp.setIncludeIgnorableWhitespace(false);
        domp.parse(new InputSource(flujo));
        doc = domp.getDocument();
        buildProcDocIterator();
      }catch(IOException e){System.out.println(e);}
  }
  private void buildProcDocIterator(){
    LinkedList listaInserts = new LinkedList();
    int tipo;
    String cadena;
    String nombreTabla;
```

```
    Element raiz;
    raiz = doc.getDocumentElement();
    nombreTabla = raiz.getNodeName();
    NodeList sentencias = raiz.getChildNodes();
    int cantSent = sentencias.getLength();
    for(int i = 0 ; i < cantSent ; i++){
      ElementImpl temp = (ElementImpl)sentencias.item(i);
      String nomSent = temp.getNodeName();
      if(nomSent.equals("insert")) tipo = 0;
      else if(nomSent.equals("delete")) tipo = 1;
      else tipo = 2;
      switch(tipo){
        case 0:
          SentInsert inse = new SentInsert(temp,nombreTabla,tipos);
          cadena = inse.getSent();
          break;
        case 1:
          SentDelete dele = new SentDelete(temp,nombreTabla,tipos);
          cadena = dele.getSent();
          break;
        case 2:
          SentUpdate upda = new SentUpdate(temp,nombreTabla,tipos);
          cadena = upda.getSent();
          break;
        default:
          cadena = new String("");
      }
      listaInserts.add(cadena);
    }
    ite = listaInserts.iterator();
  }
  /**
    Retorna un Iterador con las sentencias
  */
  public Iterator getIterator(){
    return ite;
  }
}
```

Clase Raiz

```
package moduloxmlsqlmx;

/**
  Objeto raiz para el documento XML
  y el documento DTD.
*/

public class Raiz{
  /**
    Nombre de la raiz.
  */
  private String name;

  /**
    Crea un objeto raiz con el nombre "resultset".
  */
  public Raiz(){
    name = new String("resultset");
  }

  /**
    Crea un objeto raiz con el nombre name.
  */
  public Raiz(String name){
    this.name = name;
  }

  /**
    Retorna el nombre de la raiz.
  */
  public String getName(){
    return name;
  }
}
```

Clase SentDelete

```
package moduloxmlsqlmx;
import org.w3c.dom.*;
import org.apache.xerces.dom.*;

/**
  Clase que genera una sentencia delete a partir de un elemento DOM
*/

class SentDelete{
  private String cadena;

  /**
    Crea un objeto SentDelete
  */
  public SentDelete(ElementImpl temp, String nombreTabla, DocTypesX tiposX){
    NodeList sentenciaE = temp.getChildNodes();
    ElementImpl whereEti;
    cadena = new String("DELETE FROM " + nombreTabla);
    int cantCampos = sentenciaE.getLength();
    if(cantCampos > 0){
      whereEti = (ElementImpl)sentenciaE.item(0);
      if(whereEti.getNodeName().equals("sentencia"))
        cadena = cadena.concat(" WHERE " + imprimeSent(whereEti,tiposX));
      else
          cadena = cadena.concat(" WHERE " + imprimeParen(whereEti,tiposX));
    }
  }

  private String imprimeSent(ElementImpl whereEti, DocTypesX tiposX){
    String cadenaTemp = whereEti.getAttribute("comp");
    whereEti = (ElementImpl)whereEti.getFirstChild();
    String cad = whereEti.getNodeName();
    TextImpl ti = (TextImpl)whereEti.getFirstChild();
    if(ti != null){
      String tipoCamp = tiposX.getType(whereEti.getNodeName());
      DataTypeX tip = new DataTypeX(tipoCamp);
```

```
    if(tip.isChar())
      cad = cad.concat(cadenaTemp + "'" + ti.getNodeValue() + "'");
    else if(tip.isNumeric())
      cad = cad.concat(cadenaTemp + ti.getNodeValue());
    else if(tip.isDate())
      cad = cad.concat(cadenaTemp + "\"" + ti.getNodeValue() + "\"");
    else
      cad = cad.concat(" IS NULL");
  }
  else{
    String tipoCamp = tiposX.getType(whereEti.getNodeName());
    DataTypeX tip = new DataTypeX(tipoCamp);
    if(tip.isChar())
      cad = cad.concat(cadenaTemp + "''");
    else
      cad = cad.concat(" IS NULL");
  }
  return cad;
}

private String imprimeParen(ElementImpl whereEti, DocTypesX tiposX){
  String cadenaTemp = whereEti.getAttribute("conj");
  ElementImpl izq = (ElementImpl)whereEti.getFirstChild();
  ElementImpl der = (ElementImpl)whereEti.getLastChild();
  String cad = "(";
  if(izq.getNodeName().equals("sentencia"))
    cad = cad.concat(imprimeSent(izq,tiposX) + " " + cadenaTemp + " ");
  else
    cad = cad.concat(imprimeParen(izq,tiposX) + " " + cadenaTemp + " ");
  if(der.getNodeName().equals("sentencia"))
    cad = cad.concat(imprimeSent(der,tiposX) + ")");
  else
    cad = cad.concat(imprimeParen(der,tiposX) + ")");
  return cad;
}

/**
  Retorna una sentencia
```

```
  */
  public String getSent(){
    return cadena;
  }
}
```

Clase SentInsert

```
package moduloxmlsqlmx;
import org.w3c.dom.*;
import org.apache.xerces.dom.*;

/**
  Clase que genera una sentencia insert a partir de un elemento DOM
*/

class SentInsert{
  private String cadena;

  /**
    Crea un objeto SentInsert
  */
  public SentInsert(ElementImpl temp, String nombreTabla, DocTypesX tiposX){
    NodeList campos = temp.getChildNodes();
    int cantCampos = campos.getLength();
    String[] tipos = new String[cantCampos];
    cadena = new String("INSERT INTO " + nombreTabla + " (");
    for(int j = 0 ; j < cantCampos ; j++){
        ElementImpl camp = (ElementImpl)campos.item(j);
        String nombreNodo = camp.getNodeName();
        String tipoNodo = tiposX.getType(nombreNodo);
        tipos[j] = tipoNodo;
        if(j != 0) cadena = cadena.concat(",");
        cadena = cadena.concat(nombreNodo);
    }
    cadena = cadena.concat(") VALUES (");
    for(int j = 0 ; j < cantCampos ; j++){
      Element camp = (Element)campos.item(j);
      if(j != 0) cadena = cadena.concat(",");
      TextImpl ti = (TextImpl)camp.getFirstChild();
      if(ti != null){
        String tipoCamp = tipos[j];
        DataTypeX tip = new DataTypeX(tipoCamp);
        if(tip.isChar())
          cadena = cadena.concat("'" + ti.getNodeValue() + "'");
```

```
        else if(tip.isNumeric())
          cadena = cadena.concat(ti.getNodeValue());
        else if(tip.isDate())
          cadena = cadena.concat("\"" + ti.getNodeValue() + "\"");
        else
          cadena = cadena.concat("null");
      }
      else{
        String tipoCamp = tipos[j];
        DataTypeX tip = new DataTypeX(tipoCamp);
        if(tip.isChar())
          cadena = cadena.concat("''");
        else
          cadena = cadena.concat("null");
      }
    }
    cadena = cadena.concat(")");
  }

  /**
    Retorna una sentencia
  */
  public String getSent(){
    return cadena;
  }
}
```

Clase SentUpdate

```
package moduloxmlsqlmx;

import org.w3c.dom.*;
import org.apache.xerces.dom.*;

/**
  Clase que genera una sentencia update a partir de un elemento DOM
*/

class SentUpdate{
  private String cadena;

  /**
    Crea un objeto SentUpdate
  */
  public SentUpdate(ElementImpl temp, String nombreTabla, DocTypesX tiposX){
    ElementImpl setEti = (ElementImpl)temp.getFirstChild();
    ElementImpl whereEti = (ElementImpl)temp.getLastChild();
    String cadenaTemp;

    NodeList campos = setEti.getChildNodes();
    int cantCampos = campos.getLength();
    cadena = new String("UPDATE " + nombreTabla + " SET ");
    for(int j = 0 ; j < cantCampos ; j++){
      ElementImpl camp = (ElementImpl)campos.item(j);
      if(j != 0) cadena = cadena.concat(",");
      cadenaTemp = camp.getNodeName();
      if(cadenaTemp.equals("condicion")){
        cadenaTemp = camp.getAttribute("fn");
        camp = (ElementImpl)camp.getFirstChild();
        cadena = cadena.concat(camp.getNodeName() + "="
                 + camp.getNodeName() + cadenaTemp);
      }
      else
        cadena = cadena.concat(camp.getNodeName() + "=");
      TextImpl ti = (TextImpl)camp.getFirstChild();
      if(ti != null){
        String tipoCamp = tiposX.getType(camp.getNodeName());
        DataTypeX tip = new DataTypeX(tipoCamp);
        if(tip.isChar())
          cadena = cadena.concat("'" + ti.getNodeValue() + "'");
        else if(tip.isNumeric())
          cadena = cadena.concat(ti.getNodeValue());
        else if(tip.isDate())
          cadena = cadena.concat("\"" + ti.getNodeValue() + "\"");
        else
          cadena = cadena.concat("NULL");
      }
      else{
        String tipoCamp = tiposX.getType(camp.getNodeName());
        DataTypeX tip = new DataTypeX(tipoCamp);
        if(tip.isChar())
          cadena = cadena.concat("''");
        else
          cadena = cadena.concat("NULL");
      }
    }

    if(whereEti.getNodeName().equals("sentencia")){
```

```
      cadena = cadena.concat(" WHERE " + imprimeSent(whereEti,tiposX));
    }
    else{
      if(whereEti.getNodeName().equals("parentesis")){
        cadena = cadena.concat(" WHERE " + imprimeParen(whereEti,tiposX));
      }
    }
  }

  private String imprimeSent(ElementImpl whereEti, DocTypesX tiposX){
    String cadenaTemp = whereEti.getAttribute("comp");
    whereEti = (ElementImpl)whereEti.getFirstChild();
    String cad = whereEti.getNodeName();
    TextImpl ti = (TextImpl)whereEti.getFirstChild();
    if(ti != null){
      String tipoCamp = tiposX.getType(whereEti.getNodeName());
      DataTypeX tip = new DataTypeX(tipoCamp);
      if(tip.isChar())
        cad = cad.concat(cadenaTemp + "'" + ti.getNodeValue() + "'");
      else if(tip.isNumeric())
        cad = cad.concat(cadenaTemp + ti.getNodeValue());
      else if(tip.isDate())
        cad = cad.concat(cadenaTemp + "\"" + ti.getNodeValue() + "\"");
      else
        cad = cad.concat(" IS NULL");
    }
    else{
      String tipoCamp = tiposX.getType(whereEti.getNodeName());
      DataTypeX tip = new DataTypeX(tipoCamp);
      if(tip.isChar())
        cad = cad.concat(cadenaTemp + "''");
      else
        cad = cad.concat(" IS NULL");
    }
    return cad;
  }

  private String imprimeParen(ElementImpl whereEti, DocTypesX tiposX){
    String cadenaTemp = whereEti.getAttribute("conj");
    ElementImpl izq = (ElementImpl)whereEti.getFirstChild();
    ElementImpl der = (ElementImpl)whereEti.getLastChild();
    String cad = "(";
    if(izq.getNodeName().equals("sentencia"))
      cad = cad.concat(imprimeSent(izq,tiposX) + " " + cadenaTemp + " ");
    else
      cad = cad.concat(imprimeParen(izq,tiposX) + " " + cadenaTemp + " ");
    if(der.getNodeName().equals("sentencia"))
      cad = cad.concat(imprimeSent(der,tiposX) + ")");
    else
      cad = cad.concat(imprimeParen(der,tiposX) + ")");
    return cad;
  }

  /**
    Retorna una sentencia
  */
  public String getSent(){
    return cadena;
  }
}
```

Anexo E

Glosario

applet.- Pequeña aplicación Java que se ejecuta en un servidor web y que se envía al usuario junto a una página web con objeto de realizar determinadas funciones, tales como el acceso a bases de datos o la personalización de dicha páginas web.

ANSI (*American National Standards Institute*, Instituto de Nacional Americano de Normalización)**.-** Esta organización es responsable de proporcionar estándares americanos es muchas áreas, incluyendo computación y comunicaciones. Los estándares aprobados por esta organización se conocen con frecuencia como estándares ANSI.

API (*Applications Programming Interface*, Interfaz de Programación de Aplicaciones)**.-** Es un conjunto de llamadas a convenciones las cuales definen cómo invocar un servicio a través de un paquete de software.

aplicación.- Un programa informático que lleva a cabo una función con el objeto de ayudar a un usuario a realizar una determinada actividad. WWW, FTP, correo electrónico y Telnet son ejemplos de aplicaciones en el ámbito de Internet.

ASP (*Active Server Page*, Página de Servidor Activo)**.-** Una página ASP es un tipo especial de página HTML que contiene unos pequeños programas (también llamados scripts) que son ejecutados en servidores Microsoft Internet Information Server antes de ser enviados al usuario para su visualización en forma de página HTML. Habitualmente esos programas realizan consultas a bases de datos y los resultados de esas consultas determinan la información que se envía a cada usuario específico. Los ficheros de este tipo llevan el sufijo *.asp.*

ASCII (*American Standard Code for Information Interchange*, Estándar Americano de Codificación para el Intercambio de Información)**.-** Es el formato más común para archivos de texto en computadoras. En un archivo ASCII, cada carácter alfabético, numérico o especial es representado con número binario de 7 bits.

cliente.- Un sistema o proceso que solicita a otro sistema o proceso que le preste un servicio. Una estación de trabajo que solicita el contenido de un fichero a un servidor de ficheros es un cliente de este servidor.

columnas.- En el modelo relacional, **c**uando se colocan varias filas de una tabla una debajo de otra, sus valores quedan alineados verticalmente, formando columnas. A cada columna se le asigna un nombre que la identifica y distingue de las otras de la misma tabla. Los datos que contiene una columna son homogéneos [26].

COM (*Component Object Model*, Modelo de Objetos Componentes)**.-** Es un estándar y una arquitectura que administra la reutilización binaria de componentes [29].

componente.- Es un término que representa el archivo en el que se implementan las diferentes clases del Modelo de Objetos Componentes (COM). El componente contiene, el código de creación de cada una de las clases que lo forman, así, cada clase se incorpora a un componente (se pueden implementar varias clases) [29].

dato.- Unidad mínima entre las que componen una información. Es una palabra latina que significa "lo que se da" y que apenas se utiliza en inglés, donde se suele utilizar "data" tanto para el singular como para el plural.

DTD (*Document Type Definition*, Definición de Tipo de Documento)**.-** Es el conjunto de normas XML para la representación de documentos de un determinado tipo. En ella se describe la composición de cada elemento que puede aparecer en un documento [20].

DOM (*Document Object Model*, Modelo de Objeto Documento)**.-** Es una API que proporciona un estructura de árbol de objetos. Permite manipular la jerarquía de esos objetos. DOM es ideal para aplicaciones interactivas porque el modelo de objetos esta presente en memoria, donde puede ser accesado y manipulado por el usuario.

esquema.- El diseño de la estructura que constituye una tabla o base de datos se conoce como esquema. Un esquema casi nunca se modifica.

filas.- Grupo de datos que se alinean horizontalmente, uno al lado del otro, formando un renglón o hilera. Una fila se conoce también como registro [26].

GML (*Generalized Markup Language*, Lenguaje Generalizado de Marcas)**.-** Permite la edición y formateo de texto, además del desarrollo de sistemas de recuperación de información para compartir documentos. GML introdujo el concepto de tipo de documento formalmente definido con un a estructura explícita de elementos anidados.

HTML (*Hyper Text Markup Language*, Lenguaje de Marcas de Hipertexto)**.-** Es una aplicación de SGML y es utilizado como el lenguaje estándar de publicación en Internet.

HTTP (*Hypertext Transfer Protocol*, Protocolo de Transferencia de Hipertexto)**.-** Permite que los sistemas sean construidos independientemente de los datos

transferidos, y es el más usado para navegar en Internet. La transferencia de hipertexto es simplemente la transferencia de archivos hipertexto de un computador a otro. El protocolo de transferencia de hipertexto es el conjunto de reglas utilizadas por las computadoras para transferir archivos de hipertexto, páginas web por Internet.

interfaz (*interfase* , interface)**.-** Zona de contacto, conexión entre dos componentes de "hardware", entre dos aplicaciones o entre un usuario y una aplicación.

Internet .-Red de telecomunicaciones nacida en 1969 en los EE.UU. a la cual están conectadas centenares de millones de personas, organismos y empresas en todo el mundo, mayoritariamente en los países más desarrollados, y cuyo rápido desarrollo está teniendo importantes efectos sociales, económicos y culturales, convirtiéndose de esta manera en uno de los medios más influyentes de la llamada Sociedad de la Información y en la Autopista de la Información por excelencia. Fue conocida como ARPANET hasta 1974.

Internet Explorer.- Navegador WWW creado por la empresa norteamericana Microsoft. Es uno de los navegadores Internet más difundidos.

interoperabilidad.- Es esencial en los ambientes heterogéneos actuales. Comienza con los protocolos de redes y las medidas de seguridad y se extiende hasta las redes, las operaciones heterogéneas distribuidas y la administración de los sistemas. Como punto intermedio se encuentran el acceso a los datos, la migración de aplicaciones y el acceso a las mismas en diferentes plataformas.

ISO (*International Organization for Standardization*, Organización Internacional para la Estandarización)**.-** Organización voluntaria fundada en 1946 responsable de crear los estándares internacionales en muchas áreas, incluyendo computación y electrónica. Sus miembros son las organizaciones nacionales de estándares de 89 países, incluyendo ANSI de Estados Unidos.

Java.- Lenguaje de programación desarrollado por la empresa Sun para la elaboración de pequeñas aplicaciones exportables a la red (applets) y capaces de operar sobre cualquier plataforma a través, normalmente, de navegadores WWW. Permite dar dinamismo a las páginas Web.

JavaScript.- Lenguaje de programación para WWW desarrollado por Netscape. Al igual que VBScript, pertenece a la familia Java pero se diferencia de este último en que los programas están incorporados en el fichero HTML.

JSP (*Java Server Page*, Página de Servidor Java)**.-** Una página JSP es un tipo especial de página HTML que contiene unos pequeños programas (también llamados scripts) que son ejecutados en servidores Netscape antes de ser enviados al usuario para su visualización en forma de página HTML. Habitualmente esos programas realizan consultas a bases de datos y los resultados de esas consultas determinan la

información personalizada que se envía a cada usuario específico. Los ficheros de este tipo llevan el sufijo *.jsp*

navegador.- Aplicación para visualizar todo tipo de información y navegar por el espacio Internet. En su forma más básica son aplicaciones hipertexto que facilitan la navegación por los servidores de información Internet; cuentan con funcionalidades plenamente multimedia y permiten indistintamente la navegación por servidores WWW, FTP, Gopher, el acceso a grupos de noticias, la gestión del correo electrónico, etc.

navegador Netscape.- Navegador WWW creado en 1995 por Marc Andreessen, de la empresa norteamericana Netscape. Es uno de los navegadores Internet más difundidos.

metadatos.- Datos que proporcionan información acerca de los datos de una base de datos.

modelo cliente-servidor.- Modelo de comunicación entre ordenadores conectados a una red en el cual hay uno, llamado servidor, que satisface las peticiones realizadas por otro llamado cliente.

OASIS (*Organization for Advancement of Structured Information System*s, Organización para el Avance de Sistemas de Información Estructurada)**.-** Es una organización internacional sin fines de lucro, que especifica estándares de diseño y desarrollo para la industria para la interoperabilidad basada en XML.

OLE. (*Object Linking and Embedding*, Vinculación e Incrustación de Objetos)**.-** Es el mecanismo utilizado para crear documentos compuestos. Se basa en COM. Por ejemplo, para insertar un libro de Microsoft Excel en un documento de Microsoft Word se utiliza OLE [29].

protocolo.- Descripción formal de formatos de mensaje y de reglas que dos ordenadores deben seguir para intercambiar dichos mensajes. Un protocolo puede describir detalles de bajo nivel de las interfaces máquina-a-máquina o intercambios de alto nivel entre programas de asignación de recursos.

registro.- Véase Fila.

SAX.- Es un API para XML, es una interfaz estándar para analizar eventos basados en XML, fue creada con la colaboración de miembros del XML-DEV y evaluada por OASIS. Proporciona un mecanismo orientado a eventos y de acceso serial para XML, realizando un procesamiento elemento por elemento. SAX lee y escribe XML a un repositorio de datos o a Internet.

scripts.- Código de un lenguaje de programación que se agrega a una página HTML y que se ejecuta del lado del cliente[30].

servlet.- Son piezas de código escrito en JAVA que se ejecutan en un servidor web y le añaden funcionalidad del mismo modo que un applet le añade funcionalidad a un navegador (pero no tienen interfaz gráfica). Están diseñados para soportar un modelo de pregunta/respuesta que es el más usado en el mundo web (el cliente hace una petición y el servidor le manda un mensaje de respuesta).

SGML (*Standard Generalized Markup Language*, Lenguaje Estándar de Marcas Generalizado)**.-** Es un lenguaje para describir lenguajes de marcas, particularmente aquellos utilizados en el intercambio, administración y publicación de documentos electrónicos.

sockets.- Es el identificador único hacia o desde donde la información es transmitida en una red. El socket es un número de 32 bits. Los números pares identifican a los números receptores y los números impares a los transmisores.

SQL (*Structured Query Language*, Lenguaje Estructurado de Consulta)**.-** Se utiliza para pedir a un sistema administrador de bases de datos que realice operaciones de consulta, modificación y control sobre las tablas de una base de datos [26].

tabla.- Disposición rectangular de datos en filas y columnas, en el modelo relacional una tabla representa un conjunto de entidades de un solo tipo [26].

tipo de documento.- Se refiere al vocabulario como a las restricciones para su uso. Está definido por sus elementos. Si dos documentos contienen elementos totalmente distintos o permiten muy diversas combinaciones de elementos, entonces es posible que no se ajusten a un mismo tipo de documento [20].

UML(*Unified Model Language*, Lenguaje Unificado de Modelado)**.-** Es un lenguaje de modelado que permite hacer análisis y diseño de sistemas, valiéndose de diagramas que muestran la estructura y comportamiento de los objetos que los componen.

UNICODE.- Es un estándar que proporciona un número único para cada carácter sin importar la plataforma, el programa o el lenguaje que se utilice.

W3C (*W3 Consortium*, Consorcio W3)**.-** Organización apadrinada por el MIT y el CERN, entre otros, cuya misión es el establecimiento de los estándares relacionados con WWW. Fue promovida por el creador del WWW, Tim Berners-Lee.

XML (*eXtensible Markup Language*, Lenguaje de Marcas Extensible)**.-** Lenguaje desarrollado por el W3 Consortium para permitir la descripción de información contenida en el WWW a través de estándares y formatos comunes, de manera que

tanto los usuarios de Internet como programas específicos (agentes) puedan buscar, comparar y compartir información en la red. El formato de XML es muy parecido al del HTML aunque no es una extensión ni un componente de éste.

Bibliografía

[1] Abraham Silberschatz. **Fundamentos de bases de Datos**. Mc. Graw Hill.

[2] klaine, Kevin.**SQL in a Nutshell**. O'Reilly, 2000

[3] Robert Signore. **The ODBC Solution**. Mc. Graw Hill, 1995

[4] George Reese.**Database programming with JDBC an d JAVA.** O'Reilly, 1997.

[5] **Acceso a base de datos OLEDB.** http://www.microsoft.com/data/oledb/default.htm

[6] **Oracle Utilería XSU**. www.oracle.com

[7] **XMLSQL SQL** Server www.microsoft.com

[8] SABD **SQLmx Gramática** http://148.204.20.13:8080/public_html/paginas/index.html

[9] El SABD **SQLmx**.Documentación JDBC. http://dijkstra.cic.ipn.mx:8080/public_html/paginas/documento/CaractJDBC.html

[10] Maruyama, Tamuta, Uramoto. **Creación de sitios web con XML y java**. Prentice Hall,2000.

[11] Benoît Marchal. **XML con ejemplos**. Prentice Hall, 2001.

[12] Guía del programador de aplicaciones Release 1 (9.0.1) **Utilería XSU Oracle** http://download-west.oracle.com/otndoc/oracle9i/901_doc/appdev.901/a88894/adx07xsu.htm#1000433

[13] El **Modelo de Objeto de Document**o (DOM) del W3C. http://www.w3.org/DOM/Activity.html

[14] Hunter, David.**Beginning XML**. Wrox Press.

[15] **API SAX** www.magginson.com

[16] Organización Apache **Proyecto Jakarta-TomCat**, www.apache.org

[17] Request for Comments: **RFC-2183** http://www.faqs.org/rfcs/rfc2388.html

[18] Elmasri y Navathe.**Fundamentals of Database Systems**. Addison-Wesley,1994.

[19] Flores, Ivan. **Database Architecture**. Van Nostrand Reinhold Company, 1981.

[20] Goldfarb, Charles F. y Prescod,Paul. **Manual de XML**. Prentice Hall, 1999.

[21] Levethal, Michael y Lewis, David. **Designing XML Inernet Applications**. Prentice Hall, 1998.

[22] Nakhimovsky, Alexander. **Java XML programming**. Wrox, 1999.

[23] Simpson, John E. **Just XML**. Prentice Hall, 1999.

[24] Analizador XML **Xerces**. www.apache.org

[25] Recomendación W3C 10 de febrero de 1998. **Extensible Markup Language (XML) 1.0** http://www.w3.org

[26] Benavides Abajo J. **SQL para usuarios y programadores.** Paraninfo, 1991.

[27] Bruce Eckel. **Thinking in Java.** Release 12

[28] Jack Sturm. **Desarrollo de soluciones XML**.Mc. Graw Hill

[29] Date, C.J. **An Introduction to Database Systems**. Addison-Wesley,1995

[30] Ducharme, Bob.**XML The Annotated Specification**.Prentice Hall,1999.

[31] Javier García de Jalón. **Manual Aprenda Servlets.** Universidad de Navarra.

[32] Michael J. Young. **Step by Step XML.** Microsoft Press,2000.