Ciclo Formativo DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Módulo 6 Acceso a Datos

Unidad Formativa 3

INICIATIVA Y COORDINACIÓN
IFP Innovación en Formación Profesional

Supervisión editorial y metodológica:
Departamento de Producto de Planeta Formación

Supervisión técnica y pedagógica:
Departamento de Enseñanza de **IFP** Innovación en Formación Profesional

Módulo: Acceso a Datos
UF 3 / Desarrollo de Aplicaciones Multiplataforma

Barcelona (España), 2017

MÓDULO 6

Unidad Formativa 3

Acceso a Datos

Esquema de contenido

OBJETIVOS

- Valorar las ventajas e inconvenientes de utilizar una base de datos nativa XML.
- Instalar y configurar un sistema gestor de base de datos.
- Establecer la conexión con una base de datos XML.
- Desarrollar aplicaciones que efectúen consultas sobre el contenido de la base de datos XML.
- Añadir y elminar conexiones de la base de datos XML
- Desarrollar apliaciones para añadir, modificar y eliminar documentos XML de la base de datos.

INTRODUCCIÓN

Con la aparición de XML, como recomendación del consorcio W3C en 1998, se establecieron los cimientos de lo que actualmente se llaman "aplicaciones web". XML arrastra tras de sí un conjunto de lenguajes y herramientas que facilitan la integración de datos XML y la interacción entre los diferentes niveles que definen una aplicación. El nivel más bajo, el de la persistencia, no ha escapado de la influencia de XML, por lo que se han desarrollado en los últimos años sistemas gestores XML nativos (y extensiones), resurrección en muchos casos de los anteriores sistemas gestores para datos semiestructurados (como Lore), que han desafiado al todopoderoso sistema relacional y lo han desbancado como única alternativa posible para lograr la persistencia de cualquier desarrollo software.

En este capítulo se tratarán los sistemas gestores XML nativos, se compararán con los sistemas relacionales y se desarrollarán alternativas para consultar, modificar y acceder a los datos a través de API específicas.

1. BASES DE DATOS NATIVAS XML, COMPARATIVA CON BASES DE DATOS RELACIONALES

Como ya se ha comentado en capítulos anteriores, XML (Extensible Markup Lenguaje) es un metalenguaje que permite definir lenguajes estructurados basados en etiquetas. XML es un estándar definido por el W3C que ofrece muchas ventajas como su extensibilidad, ser fácil de leer, ser portable entre diferentes arquitecturas, etc.

Por toda su potencialidad, XML ha sido ampliamente aceptado como formato de intercambio de información, por ejemplo, para definir la configuración de un sistema o para representar la estructura de un archivo. Sin embargo, en este capítulo se muestra XML como un contenedor de información en sí, es decir como una base de datos que contiene información susceptible de ser consultada.

Desde este punto de vista, un archivo XML que contenga información sobre los contactos de un usuario en una aplicación de correo electrónico se puede decir que es una base de datos, ya que sobre él se puede consultar, por ejemplo, cuántos usuarios hay en la libreta de direcciones o la dirección de e-mail de todos aquellos cuyo apellido comienza por R. Además, sobre esa base de datos se pueden añadir nuevas entradas a la libreta de direcciones, se pueden modificar entradas existentes e incluso se pueden borrar. Por tanto, un archivo XML se puede interpretar como una base de datos, y en ese caso las operaciones que se realicen sobre ella deben ser procesadas eficientemente.

Definir un sistema gestor específico para XML lleva consigo una adaptación propia en su interior, respecto a la forma en la que se estructuran, indexan y almacenan los datos. El modelo de datos XML es muy diferente a otros modelos subyacentes en sistemas gestores más tradicionales (como el modelo relacional o el de objetos). Uno de los aspectos más característicos del modelo de datos XML es su característica de semiestructura de datos, que se contrapone a las estructuras de datos características del modelo relacional.

El modelo relacional es lo que se conoce como una estructura fija de datos. Cuando se quiere diseñar una base de datos basada en el modelo relacional es necesario definir a priori qué estructura tendrán esos datos: si se dividirán en tablas (relaciones) y dentro de esas tablas si habrá atributos que concretan el tipo de los datos que se registrarán en sus instancias. Así, por ejemplo, para diseñar un sistema de correo electrónico según un modelo relacional habrá una tabla llamada contactos con atributos como el nombre, la dirección y el teléfono, por ejemplo. El modelo relacional exige que, además de establecer relaciones explícitas entre las tablas, los atributos tengan un tipo fijo definido y que los datos que sean instancias de esos atributos respeten estrictamente ese tipo. Así, un nombre de un contacto se puede definir como tipo varchar(10) lo que significa que solo podrá contener caracteres alfanuméricos con longitud menor o igual a 10.

En un modelo relacional no se permitirá que los datos que se almacenen según su estructura no cumplan a raja tabla todas sus restricciones. Sin embargo, en un modelo de datos XML esta estructura no es tan estricta, es decir, más que una estructura de datos es una semiestructura. A modo de ejemplo, en un modelo de datos XML el tipo de los datos puede ser definido después de haber creado los datos (el documento XML en sí). De hecho, el tipo de un dato puede cumplir solamente parte de la estructura de los datos almacenados y no satisfacer la de otros. De la misma manera, la estructura del esquema también puede definirse después de haberse creado los datos.

Esta flexibilidad que permiten las semiestructuras en general (y XML en particular) tiene importantes consecuencias cuando se consideran estos documentos como bases de datos. Una de estas consecuencias es la manera de expresar las consultas. En los modelos relacionales se utiliza el lenguaje SQL para hacer consultas. Ya que el modelo relacionales una estructura estricta que tiene que ser definida antes de que los datos se creen, es viable que SQL obligue al usuario a conocer esa estructura para poder expresar una

consulta. Así, una consulta SQL para obtener información sobre los ejemplares que hay del libro Crimen y castigo en una librería se haría de la siguiente forma:

Select: l.titulo, e. edicion From libro 1, ejemplar e

Where l.isbn=e.ison and l. titulo like "3 Crimen y castigo:"

En esta consulta sobre un modelo relacional se sabe que hay una tabla libro que tienen un atributo titulo que es de tipo alfanumérico. Además, hay una tabla ejemplar que está relacionada con libro mediante el isbn. Si no se supiese la estructura relacional para esta base de datos, no se podría hacer esta consulta debido a lo estricto de la definición del modelo de datos relacional y del álgebra del propio lenguaje de consulta SQL.

Con las características de semiestructura de datos de XML, SQL no podría nunca ser un lenguaje apropiado para consultar este modelo ya que la estructura exacta de un documento XML no tiene por qué conocerse a priori, ni siquiera se puede garantizar que todos los datos del documento XML sigan una única estructura. Por ejemplo: un documento XML para una librería puede tener un elemento <libro> y parte de los elementos libro tienen definido su título como un elemento hijo llamado <titulo> sin embargo, y aquí viene el matiz, otros elementos <libros definen su título como un atributo XML. El siguiente XML muestra un ejemplo.

Ejemplo

```
<?xml version="1.0" encoding="UTF-8" 2x
<Libros> <Libro> <Autor:>Dolaney, Kalen</Autor> «Titulo»Inside Microsoft SQL Server 2000</Titulo» </Libro>
<Libro>
<Autor> Burton, Kevin</Autor.> <Titulo>.NET Common Language Runtime.</Titulo> </Librox
<Libro titulo= "C# Design Patterns"> <Autor>Cooper, James W. </Autori> </Librox
</Libro>
```

Para un modelo de s como el de XML, el álgebra del lenguaje de consulta que se utilice no puede ser tan estricta como SQL, sino que debe teneralternativas que permitan flexibilizar la estructura que siguen los datos (y que puede ser casi desconocida). Lenguajes de consulta para XML como XPath oXQuery siguen álgebras más adecuadas y operadores que dan más flexibilidad a las consultas que SQL.

A modo de resumen de lo expuesto, en el ámbito de los sistemas gestores de bases de datos la flexibilidad de XML afecta a la concepción integra del sistema, a todas las piezas que hacen que un sistema gestor almacene y consulte datos eficientemente tanto en tiempo como en número de recursos necesarios. El modelo de datos XML no solo afecta a la forma en la que se expresan las consultas, sino que también toca otros aspectos básicos en el diseño de sistemas gestores como es la representación interna del modelo de datos, la indexación de contenidos o el control de transacciones, por nombrar algunos.

1.1. DOCUMENTOS CETNRADOS EN DATOS Y EN CONTENIDO

Para profundizar en el modelo XML y su visión dentro de un sistema gestor, en esta sección se definen dos puntos de vista de un documento XML, dos modelos diferentes que condicionan la estructura interna de los sistemas gestores XML. El primero, documento centrado en datos, trata un documento XML como

una estructura fija de datos tal y como lo haría un modelo relacional, el segundo, documento centrado en contenido, lo trata más como lo que es, una semiestructura de datos.

El modelo de documento centrado en datos (data-centric) se suele emplear cuando se usa XML como documento para el intercambio de datos. Suelen ser documentos con estructuras regulares y bien definidas. Es una visión de un XML más como un modelo estricto similar al relacional o al de objetos.

El modelo de documento centrado en el contenido (document-centric) usa un documento XML como lo que es, una estructura no estricta e irregular (semiestructura), explotando todas sus posibilidades.

Estos dos modelos son extremos y es difícil ubicar un documento en particular dentro de uno de los dos. La clasificación de documentos no es siempre directa y clara, y en múltiples ocasiones el contenido estará mezclado o será difícil de catalogar en un tipo u otro: se puede tener un documento centrado en datos donde uno de los datos sea una parte de codificación libre, o un documento centrado en el contenido con una parte regular con formato estricto y bien definido. Sin embargo, el interés de querer clasificar los documentos en estos dos conjuntos radica en que, según se use uno u otro, se condiciona la manera en la que el sistema gestor almacenará los datos internamente y las posibilidades de consulta que pueda ofrecer. Por ejemplo, nada impide a un sistema gestor XML (que solo acepte documentos centrados en datos) que internamente los almacene como lo haría un sistema relacional, ya que este tipo de documentos obligan a los datos a seguir una estructura homogénea. Sin embargo, un sistema gestor XML que permita documentos centrados en el contenido necesita de una representación interna de los datos diferente a la que haría un sistema relacional, ya que las características de semiestructura mencionadas anteriormente no siempre pueden ser encapsuladas en una estructura tan estricta como la relacional.

1.2. ¿ALTERNATIVAS PARA ALMACENAR XML?

Una vez se han conocido los aspectos de XML que afectan a la manera de poder almacenar los datos de un documento, hay que enumerar las diferentes posibilidades de almacenamiento y así poder entender mejor qué uso se le puede dar a los documentos XML cuando son vistos como bases de datos y qué uso se le pueden dar a los sistema gestores de bases de datos dentro del desarrollo de aplicaciones.

La primera alternativa es almacenar los documentos XML en un sistema gestor relacional o de objetos. Esto es lo que se conoce como una alternativa híbrida (XML-Enabled). Es decir, en ella se usa XML como formato de trabajo en los desarrollos (representación de información, intercambio, etc.) pero luego los datos los almacena en sistemas gestores clásicos. Esta es la alternativa que utilizan actualmente, por ejemplo, los sistema gestores de contenidos (CMS) tipo Joomlal o Drupal. Estos sistemas utilizan XML como fichero intermedio de representación y configuración, pero los datos en sí son almacenados en una base de datos relacional (MySQL). Otras aplicaciones utilizan sistemas gestores relacionales para almacenar los datos, pero el resultado de las consultas lo devuelven en formato XML, luego las capas de gestión y visualización trabajan con estas salidas XML para representar los datos XML en las interfaces de usuario (por ejemplo, usando transformaciones XSL o XSLT).

No todos los documentos XML serán aptos para ajustarse a las reglas estrictas de los sistemas relacionales. De hecho, solo los documentos centrados en datos son los adecuados para mapearse en estructuras relacionales. Lo que se hace es convertir el documento XML en tablas y atributos según el modelo relacional, siguiendo un proceso llamado "mapeo". Esto se puede hacer así ya que los documentos centrados en datos tienen una estructura regular y bien definida fácilmente transformable en un esquema relacional. Sin embargo, también tiene inconvenientes ya que solo se almacenan los datos que interesa conservar, dejando en el camino otros datos propios de XML como los comentarios o el formato. De alguna manera, un documento XML recuperado de un sistema relacional en el que previamente se ha mapeado nunca se

parecerá a su original con exactitud, a no ser que el documento XML se diseñe más como se haría para un modelo relacional omitiendo y no empleando todas las posibilidades propias de XML que no se correspondan en un modelo relacional.

Dentro de la posibilidad de almacenar documentos XML en sistemas relacionales u objetuales, está la alternativa de almacenar todo el documento dentro de un campo. Por ejemplo, en Oracle (con XML DB) se puede utilizar un tipo CLOB o VARCHAR para almacenar archivos XML tratándolo así como si fuera un campo de texto, o también se puede usar un campo XType, que es una solución más específica para tratar XML, aunque esta alternativa también tiene restricciones relacionadas con qué se puede y qué no se puede almacenar del documento XML original.

La ventaja principal de esta alternativa es que deja descansar el peso de gestión de los datos en un sistema tan probado y conocido como el relacional, sin preocuparse de si otro sistema consigue o no las mismas cuotas de rendimiento y optimización que los sistemas gestores relacionales llevan consiguiendo desde hace más de 30 años. Sin embargo, esta alternativa tiene también desventajas a favor de sistemas XML nativos (que se verán a continuación):

V. Si la jerarquía de los datos XML es compleja, su conversión a un conjunto de tablas relacionales produce una gran cantidad de estas o de columnas dentro de cada tabla con valor nulo. Se pueden conectar las tablas resultantes para mantener la estructura jerárquica de XML, pero suele ser un proceso complicado para estructuras de datos XML complejas.

V Puede que se quieran hacer consultas sobre cualquier elemento o propiedad de XML, pero podría no ser posible si el elemento no está incluido en el índice de la base de datos relacional a la que se mapea.

V. Se complica la posibilidad de explotar directamente alguna tecnología relacionada conXML como, por ejemplo, las consultas XPath, XSLT, XQL, XQuery, etc.

La segunda alternativa (y la que centra el contenido de este capítulo) es utilizar sistemas gestores nativos XML. Estos sistemas aparecen debido a que el modelo de datos XML no siempre tiene una correspondencia directa con modelos más tradicionales como el relacional y a que el álgebra de los lenguajes de consulta y modificación necesita de operaciones especiales adaptadas al modelo de datos. En los últimos años han proliferado las soluciones nativas para almacenar y recuperar datos XML intentando respetar su modelo semiestructurado. Estos sistemas, como eXisto Tamino, se basan en sistemas gestores como Lore, anteriores a la aparición en 1998 de XML y que intentaban resolver el problema de los datos semiestructurados (de los que XML no es el único representante).

Estos sistemas gestores nativos XML definen mecanismos para almacenar el modelo de datos XML eficientemente, indexarlo y, por tanto, consultarlo en el menor tiempo posible y optimizando los recursos necesarios. Trabajando con este tipo de sistemas XML nativos se puede operar directamente con XML en todas las fases y capas del desarrollo, utilizarlo de manera natural tanto para las representaciones de interfaces como para el almacenamiento interno, facilitando así la homogeneidad y optimización de los desarrollos. Con los sistemas XML nativos se pueden almacenar tanto documentos centrados en los datos como centrados en el contenido sin necesidad de perder datos en mapeos previos. En resumen, un sistema XML nativo contempla los siguientes puntos:

- Almacena y recupera datos según un modelo de datos XML. Permite documentos centrados en el contenido.
- Permite las tecnologías de consulta y transformación propias de XML, (XQuery, XSLT, etc.), como vehículo principal de acceso y tratamiento.

1.3. COMPARATIVAS CON LOS SISTEMAS GESTORES RELACIONALES

Por todo lo comentado anteriormente, es evidente que hay diferencias entre los sistemas XML nativos y los sistemas relacionales. La primera diferencia es el modelo que utilizan para representar los datos internamente: el modelo XML frente al modelo relacional. Ambos modelos no son compatibles al 100% y aunque en algunos casos sea posible mapear un documento XML a un sistema relacional, la mayoría de las características XML no tienen correspondencia en un sistema relacional. Lo mismo ocurre con el álgebra de consulta que se implemente: SQL, por soportarse en un álgebra destinada al modelo relacional, nunca podrá ser un lenguaje de consulta que explote toda la potencia de XML, como lo hacen lenguajes específicos como XQuery o XPath.

Los sistemas nativos tratan al documento XML en toda su extensión contemplando todos sus elementos. Así, por ejemplo, el orden en el que aparecen los datos en un documento XML puede llegar a ser muy importante para su interpretación futura. Los sistemas XML nativos respetan ese orden y ofrecen herramientas para tratarlo. Sin embargo, si se almacena el XML en un sistema relacional, por el propio modelo relacional, ese orden se pierde y no ofrece herramientas para respetarlo (a no ser que se haga un mecanismo que permita ordenar los datos según su orden en el documento XML original).

Los sistemas XML nativos contemplan elementos, atributos, comentarios y muchos otros aspectos de XML, incluso la posibilidad de opcionalidad de elementos de la estructura, cosa que un modelo relacional no puede soportar: en un sistema relacional nunca podrá haber una tabla con un campo opcional, el campo puede tener un valor nulo o no, pero siempre debe estar en la definición de la tabla. Eso no ocurre así en un documento XML ya que es posible que un elemento sea opcional, no que no tenga valor, sino que no aparezca en la estructura o aparezca de otra manera. Un ejemplo, es el caso del título en el siguiente documento XML, el segundo libro contempla el título no como un elemento hijo, sino como un atributo. En un sistema relacional, este cambio de la estructura sería inconcebible.

Ejemplo

```
<?xml version="1.0" encoding="UTF-8" 2x <Libros> <Libro> <Autor > Burton, Kevin</Autor.>
<Titulo>.NET Common Language Runtime.</Titulo>

</Libro> <Libro titulo= "C# Design Patterns"> <Autor> Cooper, James W.</Autor> </Libro>

</Libro>
```

Con esta comparativa no se pretende hacer una labor de marketing que abogue por que se migren todas las aplicaciones existentes que funcionan muy bien con los sistemas relacionales (por ejemplo, un sistema bancario), hacia sistemas XML. Sin embargo, una conclusión que se puede extraer es que hay nuevos tipos de aplicaciones que requieren la extensibilidad de datos que ofrece XML y puede ser problemático transformarlos para su almacenamiento en tablas en un sistema relacional. En esos casos es preferible utilizar sistemas nativos XML.

Dicho esto, más allá de las ventajas cualitativas de los sistemas XML nativos para almacenar XML, es cierto que el uso de alternativas híbridas (XML-Enabled) que almacenan XML en sistemas relacionales tiene su aceptación. Aunque, como ya se ha subrayado anteriormente, convertir un modelo XML a uno relacional (o de objetos) lleva consigo una pérdida de la esencia XML, no es menos cierto que, en general, los niveles de optimización de recursos, tiempo de respuesta y escalabilidado son mejores en sistemas relacionales que en sistemas nativos XML.

No hay que olvidar que el modelo relacional es un modelo matemático muy optimizado. Este modelo permite algoritmos de optimización de consultas muy depurados y probados durante muchos años en la industria. El modelo XML, al igual que le ocurrió en su momento al modelo de objetos, no permite un nivel tan alto de optimización y unos rendimientos en generaltan potentes como los que ofrecen los sistemas gestores relacionales. Esto hace que la batalla entre los sistemas XML nativos y las alternativas híbridas soportadas por los sistemas gestores relacionales más importantes no sea una simple batalla ideológica, sino que se sustenta en el hecho de que hay aplicaciones de gestión específicas, muy críticas, en las que es determinante el efectivo control de transacciones, la optimización de consultas y el gran volumen de información que se permite almacenar y consultar. En este tipo de aplicaciones, en las que no es tan importante conservar las propiedades de semiestructura de XML como obtener rendimiento óptimo en situaciones extremas, los sistemas relacionales tienen más que demostrada su valía.

Como conclusión, un desarrollador nunca puede afirmar que un sistema gestor relacionales siempre más adecuado desde el punto de vista de rendimiento que un sistema nativo XML, ni que siempre es preferible un relacional antes un nativo XML, cada problema requiere una solución específica y es el desarrollador el que debe sopesar ventajas e inconvenientes cuando el sistema gestor se integra en toda la aplicación.

2. ESTRATEGIAS DE ALMACENAMIENTO

Como se ha comentado en los puntos anteriores, los sistemas XML nativos son una alternativa natural para aplicaciones que trabajan con documentos XML en todos sus niveles (interfaz, gestión y almacén de datos). Al igual que ocurre con otros sistemas gestores los sistemas XML nativos soportan transacciones, acceso multiusuario, lenguajes de consulta, índices, etc. Algunos ejemplos de sistemas nativos XML son:

Comerciales: eXcelon XIS, GoXML DB, Infonyte-DB, Tamino, X-Hive/DB. Código abierto: dibXML, eXist y Xindice. De investigación: Lore y Natix.

Aunque estos son solo algunos de los sistemas XML nativos que se pueden encontrar, se puede afirmar que son los más conocidos. De entre ellos Tamino, exist y Lore son exponentes destacados de sus segmentos respectivos (comerciales, código abierto y de investigación).

En general, todos los sistemas XML nativos tienen como principal misión el almacenamiento y la gestión de documentos XML y para ello implementan las siguientes características:

V XML permite asignar una estructura a documentos XML mediante esquemas XML (XML-Schema) o DTD, por lo tanto, los sistemas nativos que permitan asociar esquemas a documentos deben permitir comprobar la adecuación de los datos a esos esquemas (validación).

V. Más evidente es que los sistemas nativos deben permitir almacenar y recuperar documentos de acuerdo con la especificación XML. Como mínimo el modelo debe incluir elementos y atributos, y respetar el orden de los elementos en el documento y la estructura de árbol.

V. Para una recuperación eficiente de los datos estos deben estar indexados. El modo de indexación en este tipo de sistemas no es igual que el seguido en sistemas relacionales sino que debe ser adecuado y concreto para la estructura de datos XML.

V Como sistema gestor cualquier sistema XML nativo debe soportar concurrencia y seguridad.

V Por último, los sistemas XML nativos deben dar soporte a toda la tecnología asociada a XML.Algunos ejemplos de esta tecnología son XPath, XQuery y XSLT, en cualquiera de sus actualizaciones y versiones.

Para comprender mejor cómo funcionan los sistemas XML nativos y cómo soportan las características enumeradas anteriormente, a continuación se detallan dos sistemas concretos: Tamino, referencia dentro de los sistemas comerciales, y eXist, referencia a su vez dentro de los sistemas de código libre. Enumerar algunas de sus características permitirá entender mejor los puntos en común que comparten ellos mismos y otros sistemas XML nativos.

1. Tamino o Este sistema XML nativo es un producto comercial de Software AG. Ofrece una alternativa pura XML para almacenar y recuperar documentos ya que los almacena en una estructura propia sin necesidad de hacer uso de ningún otro sistema gestor subyacente (por ejemplo, relacional) que requiera de una transformación.

Como es habitual en este tipo de soluciones, Tamino separa los datos de los índices: por un lado define una estructura para almacenar los documentos y por otro los índices asociados que potenciarán la recuperación de datos de esos documentos.

Tamino permite almacenar documentos que siguen un esquema XML-Schema o DTD (validados) o que no lo siguen (bien formados). La segunda opción es menos potente pero da más flexibilidad.

Al igual que la gran mayoría de sistemas XML nativos Tamino estructura los documentos en colecciones. Una colección es un conjunto de documentos, de modo que forma una estructura de árbol donde cada documento pertenece a una única colección.

Dentro de las colecciones no solo se pueden almacenar documentos XML, sino que también se pueden almacenar otros tipos de documentos de texto o binarios (los llamados documentos no XML).

Los elementos de configuración del sistema también son documentos XML almacenados en la colección system, por lo que pueden ser accedidos y manipulados por las herramientas estándar proporcionadas. Es decir, todo dentro de Tamino es XML.

Tamino proporciona diferentes tipos de índices que son mantenidos automáticamente cuando los documentos son añadidos, borrados o modificados. Algunos de los índices que soportan son:

Simple text indexing: indexa palabras dentro de elementos.

o Simple Standard Indexing: indexación por valor de los elementos. o Structure Index: mantiene todos los caminos posibles según el esquema del documento.

Por último, Tamino permite diferentes maneras de acceder a los datos: servicios SOAP, API Java (XMLDB, XQJ) y API para .NET son algunas de ellas.

eXisto es una alternativa de código libre con características similares en esencia a las contempladas en Tamino. Al igual que Tamino, los documentos se almacenan en colecciones, y cada documento está en una colección. También, eXist permite que dentro de una colección puedan almacenarse documentos de cualquier tipo.

A diferencia de Tamino, en elXist los documentos no tienen que tener una DTD o XML Schema asociado, por lo que solo ofrece funcionalidad para documentos XML bien formados, sin atendera si siguen o no una estructura (validación).

El almacén central nativo de datos es el fichero dom.diba; es un fichero paginado donde se almacenan todos los nodos del documento según el modelo DOM del W3C. Dentro del mismo archivo existe también un árbol B que asocia el identificador único del nodo con su posición física. En el fichero colections.db.x se almacena la jerarquía de colecciones y relaciona esta con los documentos que contiene; se asigna unidentificador único a cada documento de la colección que es almacenado también junto al índice.

eXist automáticamente indexa todos los documentos utilizando índices numéricos para identificar los nodos del mismo (elementos, atributos, texto y comentarios). Durante la indexación se asigna unidentificador único a cada documento de la colección, el cual es almacenado también junto al índice. Para ahorrar espacio los nombres de los nodos no son utilizados para construir el índice, en su lugar se asocia el nombre del elemento y de los atributos con unos identificadores numéricos en una tabla de nombres.

Por defecto, exist indexa todos los nodos de texto y valores de atributos dividiendo el texto en palabras. En el fichero u ords.dbx se almacena esta información.

eXist proporciona diferentes maneras de acceso a datos. Puede ser usado en un servidor Java (J2EE) con servicios XML-RPC, SOAP y WebDAV, y también con APIJava (XMLDB, XQJ).

Como puede deducirse, ambos sistemas gestores tienen características muy similares. Ya que el objetivo de este capítulo es el acceso a datos almacenados en sistemas XML nativos, se hace necesario centrar el contenido siguiente en dos características esenciales que ambos sistemas en particular (y todos casi en

general) ofrecen: el modelo de colecciones y las API de acceso para aplicaciones externas. Respecto a las API de acceso, las siguientes secciones del capítulo las tratarán más extensamente. Ahora es más adecuado explicar el modelo de colecciones.

2.1. COLECCIONES Y DOCUMENTOS

Como se ha definido en la sección anterior, una colección es un conjunto de documentos, de modo que forma una estructura de árbol donde cada documento pertenece a una única colección. De alguna manera se puede afirmar que, al igual que un documento XML sigue una estructura de árbol, o los sistemas nativos utilizan esa misma estructura para organizar los documentos almacenados en ellos. Un modelo similar es un sistema de archivos de un sistema operativo donde las carpetas parten de una raíz y pueden tener subcarpetas, y es dentro de las carpetas donde se almacenan los documentos. Los sistemas nativos siguen algo parecido, ya que las colecciones pueden ser vistas como carpetas, y dentro de ellas se almacenan recursos (documentos),” los cuales pueden serXML o noXML (texto o binarios).

De db cuelgan dos colecciones:

system contiene las colecciones y documentos necesarios para la configuración de exist. Esta colección estratada por el sistema gestor, la necesita para su administración, por lo que no debe ser manejada por el usuario.

Libros ha sido creada específicamente como conjunto de datos para los ejemplos de este capítulo.

Dentro de Libros se encuentran tres colecciones hermanas: Dentro de En Castellano hay un único documento XML llamado Libros Castellano.xml. Dentro de MisFavoritos también hay un único documento XML llamado Libros Favoritos.xml

Dentro de En Ingles hay un documento y una colección hija. El documento es XML y tiene por nombre Libros Ingles.xml. La colección hija por su parte tiene por nombre Técnicos y dentro de ella hay un único documento que tiene por nombre Libros Tecnicos.xml

El ejemplo de la Figura 5.1 no contiene todos los casos que se pueden dar; por ejemplo, dentro de una colección puede haber más de un documento XML e incluso más de un documento no XML (que no aparecen en este ejemplo).

Usando los mismos términos que emplea XPath (visto en el Capítulo 1), la estructura de árbol que tienen las colecciones permite establecer caminos entre nodos hasta hacer referencia a los elementos a los que se quiere acceder o recuperar.

3. LIBRERÍAS DE ACCESO A DATOS XML

En las secciones anteriores se han expuesto las características de los sistemas XML nativos así como sus ventajas frente a otros sistemas basados en otros modelos, como por ejemplo es el sistema relacional. Además, se han descrito las particularidades de dos de los sistemas nativos más conocidos, como Tamino y eXist.

Ver en profundidad todo lo relacionado con el acceso a datos en sistemas XML nativos es una misión inabordable para un único libro. Cada sistema gestor tiene sus peculiaridades y tratarlos en profundidad requiere obras específicas y concretas para cada uno de ellos.

Por este motivo, las siguientes secciones de este capítulo se centrarán en desarrollar ejemplos prácticos sobre un único sistema gestor: eXist. Este sistema ha sido seleccionado no solo por ser una aplicación de código libre muy extendida en los entornos empresariales, sino que también por ser la mejor opción para dar los primeros pasos en este tipo de sistemas con un enfoque educativo y formativo. exist puede usarse sin restricciones y es de fácil acceso, y como añadido da soporte a todas las tecnologías básicas relacionadas con el acceso a datos XML, que han sido objeto de interés en este capítulo.

Por otro lado, lamentablemente, aun centrando los desarrollos solamente en eXist, no es posible tratar en una única obra todo lo relacionado con el acceso a datos en sistemas XML nativos desde aplicaciones o servicios externos. Actualmente, la oferta de acceso a sistemas XML nativos, al igual que ocurre en sistemas relacionales o de objetos, es muy extensa. Muchas son las técnicas, servicios y librerías que se ofertan (propias, estándares de facto o estándares reales) para poder acceder de diferentes maneras a los datos que un sistema gestor almacena.

Así, este capítulo se centrará en librerías ampliamente utilizadas en desarrollos profesionales que permiten acceder desde Java a sistemas XML nativos. Estas librerías son XML-DB y XQ.J.

XML:DB: es la librería Java de acceso a sistemas XML nativos más conocida y utilizada desde el 2001, cuando apareció su especificación. La mayoría de los sistemas gestores la soporta. El objetivo de XMLDB es la definición de un método común de acceso a sistemas nativos permitiendo la consulta, creación y modificación de contenido desde aplicaciones cliente personalizadas. XML:DB puede ser considerada una equivalencia en los sistemas XML nativos a alternativas como ODBC y JDBC. Como toda librería de acceso a datos, XMLDB está a expensas de lo que cada sistema gestor implemente de ella.

La estructura de XMLDBogira en torno a los siguientes elementos básicos:

- Driver: encapsula la lógica de acceso a una base de datos determinada. Cada sistema debe implementar este driver.
- Collection: clase que representa el concepto de colección visto en la Sección 5.2.1.
- Resource: clase que representa el concepto de recurso según se muestra en la Sección 5.2.1. Los recursos pueden ser de dos tipos:
- XMLResource: representa un documento XML o parte de un documento obtenido con una consulta. - Binary Resource: representa un documento no XML.
- Service: Implementación de una funcionalidad que extiende el núcleo de la especificación.
- XQJ (XQuery API for Java): es una propuesta más actual que XML:DB para la estandarización de acceso a sistemas XML nativos. Esta propuesta empieza en 2007, aunque no es hasta 2011

cuando los sistemas empiezan a implementarla seriamente. De alguna manera XQJ intenta asemejarse lo más posible a JDBC: especifica una estructura de clases Java con sintaxis similar a JDBC y sigue una filosofía basada en un origen de datos al que se puede conectar un cliente, y partiendo de esa conexión, lanzar peticiones de consulta.

Por sus características, y porque a diferencia de XML:DBXQJ se encuentra muy activo, esta API está llamada a ser el estándar de acceso que usarán todos los sistemas gestores XML nativos.

- Las clases más significativas de la versión XQJ 1.0o que debe implementar todo sistema gestor que quiera ser compatible con XQJ son:
- XQDataSource:identifica una fuente física de datos a partir de la cual crear conexiones; cada implementación definirá las propiedades necesarias para efectuar la conexión, siendo básicas las propiedades user y passu ord.
- XQConnection: representa una sesión con la base de datos, manteniendo información de estado, transacciones, expresiones ejecutadas y resultados. Se obtiene a través de unXQDataSource.
- XQExpression: objeto creado a partir de una conexión para la ejecución de una expresión una vez, retornando unXQResultset Sequence con los datos obtenidos. La ejecución se produce llamando al método executequery.
- XQPreparedExpression: objeto creado a partir de una conexión para la ejecución de una expresión múltiples veces, retornando unXQResultset Sequence con los datos obtenidos. Igual que en XQExpression la ejecución se produce llamando al método execute Query.
- XQResultSecuence: resultado de la ejecución de una sentencia y contiene un conjunto de 0 o más XQResultItem.

Una diferencia importante entre XQJy XMLDB es que el primero no se preocupa de las colecciones. Al igual que en una base de datos relacional, XQJ establece una conexión y luego se accederá con un lenguaje. En los sistemas gestores relacionales el lenguaje es SQL y en los sistemas XML nativos accedidos con XQJ será XQuery. De esta manera, XQJofrece un nivel mayor de abstracción que XMLDB ya que solo se preocupa de los datos almacenados en los recursos y su estructura dentro del recurso obviando la estructura de colecciones con la que se haya estructurado la base de datos.

En las siguientes secciones se trabajará con ejemplos concretos para la conexión y el acceso a un sistema XML nativo. Para ello, es necesario centrar los esfuerzos en un sistema concreto, que será eXist 1.4.2, y una implementación concreta de las API de acceso, en este caso serán XML:DB de exist y XQJ de exist.

No es objetivo del capítulo ver todas las posibilidades de ambas API, sino ver cómo solucionar escenarios básicos de acceso y esbozar las posibilidades que cada una ofrece.

3.1. CONTEXTO EN EL QUE SE USAN LAS API

El entorno utilizado en el desarrollo de los ejemplo de este libro ha sido NetBeans IDE 7.1.2 (para Windows 7) con JDK 1.7. Las librerías incluidas en los proyectos para soportar la implementación de XMLDB y XQJ en elXist han sido las siguientes:

XQ.J. Las librerías incluidas en el proyecto han sido XQJapijar; XQJ2-0.0.1jar; eXist-XQJ-1.0.1.jar y existX-QJ-examplesjar. El archivo con estas librerías se puede encontrar en exist-XQJo Para los ejemplos se ha descomprimido el archivo dentro de la carpeta instalación de eXist}/eXist/lib.

XML:DB: Las librerías necesarias que se deben incluir en el proyecto para accedera exist con esta API son:

Librerías localizadas en instalación de exist}/eXist/: eXistjar; eXist-fluentjar y exist-optional jar. Estas librerías vienen con la instalación de eXist 1.4.2.

- Librerías localizadas en instalación de eXist}/eXist/lib/core: uvs-commons-util-1.0.2.jar, xmldbjar, xmlrpc-client-3.1.2.jar, xmlrpc-common-3.1.2.jar, xmlrpc-server.3.1.2.jar, log4j-1.2.15jar. Estas librerías deben venir con la instalación de eXist 1.4.2.

Además del entorno para poder usar estas API se debe instalar en local el sistema gestor eXist. La instalación es muy sencilla. Se descarga la versión de eXist del portal." Una vez descargada, se inicia el proceso de instalación. Primero se solicitará una versión de JDK compatible (en este caso JDK 1.7) y seguidamente se solicita la ruta en la que se instalará (instalación de eXist). El último paso destacable de la instalación es que se solicita el nombre de usuario (por defecto "admin") y la passuord (en nuestro caso "admin" también) para el acceso.

Un vez instalado, se puede arrancar el servicio exist manualmente para poder acceder con la aplicación cliente. El cliente por defecto que instala exist es el exist Client Shell que, previo acceso con usuario y password, permite gestionar las colecciones y recursos de la base de datos a las que tiene acceso el usuario registrado. La Figura 5.3 muestra una captura del exist Client Shell para el ejemplo de la Sección 5.2.1.

4. ESTABLECIMIENTO Y CIERRE DE CONEXIONES

Con el establecimiento de conexiones se comienza el proceso de acceso a los datos almacenados en sistema XML nativo. En esta sección se mostrarán las funciones de conexión que ofrecen XMLDB y XQ.J. Ambas soluciones siguen el mismo proceso: cargan el driver que permite conectar a un sistema gestor en particular (en este caso eXist) y posteriormente establecen el usuario y passuvord que dan acceso a la colección en donde se encuentran los datos que se quieren gestionar (consultar, modificar, insertar o borrar). La diferencia entre XMLDB y XQJ son las funciones que utilizan para la conexión (clases), el nivel de abstracción que ofrecen respecto a la estructura de colecciones y recursos y la manera en la que implementan internamente el proceso de conexión.

4.1. CONEXIÓN CON XML:DB

Para la conexión con XML:DB se necesitan los siguientes elementos:

El driver: para exist el driver viene expresado así driver="orgeXíst.xmldb. Database Impl". Se carga el driver en memoria: c1= ClassiforMame(driver); Se crea una instancia de la base de datos y se guarda en una variable tipo Database: database - (Database) cl. new Instance () ; La instancia de la base de datos es necesario registrarla: Database Manager registerDatabase (database);

Una vez registrada ya se puede acceder a la colección deseada. La ruta de la conexión se especifica mediante una URIo (ruta de acceso al servicio que atiende la base de datos) y la ruta de la colección. La URI para una base de datos en local sería:

URI = "xmldb: exist: //localhost:8080/eXist/xmlrpc"; La ruta a una colección (por ejemplo, En Ingles) dentro de la base de datos mostrada en la Figura 5.2 sería: collection="/db/Libros/Enlingles"; Si pertenece a un usuario es necesario especificar como parámetros el usuario y la clave de acceso: usuario="admin" ; usuario Pwd='"admin";

Con todo, el acceso a la conexión sería:

Collection col = DatabaseManager.getCollection (URII collection, usuario, usuario:Pwd) ;

Con esto se obtiene en colla colección /db/Libros/En Ingles que está en la base de datos eXist cuya ruta de acceso es xmldb.eXist://localhost:8080/eXist/xmlrpc (localo y a la que solo pueden acceder el usuario y clave (admin, admin).

El siguiente código esboza todo el proceso de conexión. Como se puede observar, no se controlan los errores de conexión que puedan surgir (driver inexistente, colección inexistente, URI no válida, etc.). Si se quiere un código completo, esos posibles errores deberían ser capturados (tal y como se hará más adelante).

protected static String driver = "org.eXist.xmldb. DatabaseImpl"; public static String URI = xmldb: exist: local host:8080 / exist/xmlrpc"; private Database database;

private String usuario="admin":

private String usuario Pwd="admin"; private String collection="/db/Libros EnIngles";

Class cl - Class. for lame (driver); // Se crea un objeto Database database - (Database) Cl. new Instance (); Database Manager. registerDatabase (database);

// Se obtiene la colección (URI + collection) con el usuario y password que tiene acceso //a ella. Collection col =

DatabaseManager.get Collection (URI + Collection, usuario, usuario Pwd);

if (col.getResourceCount () == 0) { //si la colección no tiene recursos no podrá devolver ninguno

System. out.printlin ("La colección no tiene recursos");

Los parámetros usuario y password del método DatabaseManagerget Collection son opcionales ya que una colección puede pertenecer a un usuario que no haya definido una password o incluso puede estar abierta a todos.

El código obtiene en colla colección /db./Libros/En Ingles. Dentro de esa colección puede haber otras colecciones o uno o más recursos (XML o no XML). Con el método get ResourceCount0 se puede saber cuántos recursos tiene esa colección (no incluye a los que tengan los hijos). En las siguientes secciones se verá cómo accedera ellos.

4.2. CONEXIÓN CON XQJ

La conexión con XQJ es menos tediosa que en XMLDB y se asemeja más a cómo sería con un JDBC. Los pasos necesarios son:

1. Crear una fuente de datos (datasource): XOData Source xgs = new EXistXOData Source ();

Establecer las propiedades de conexión, que son básicamente el nombre del servidor y el puerto de acceso. 4. Observar que estas propiedades se han definido en XMLDB dentro de la URI.

xqs ... setProperty ("serverName", "localhost") ; xgs. set Property (oport", "8080");

3 Por último, se obtiene una conexión indicando el usuario y la clave (si es necesario) que dan permisos de acceso.

XOConnection conn = xqs.getConnection ("admin", "admin");

La clase XQConnection a la que pertenece conn tiene métodos para acceder mediante XQuery a los datos almacenados, como luego se verá.

El siguiente código esboza todo el proceso de conexión. Como se muestra, no se controlan los errores de conexión que pueden surgir (datos de conexión erróneos, usuario y passuordincorrectos, etc.). Si se quiere un código completo, esos posibles errores deberían ser capturados (tal y como se hará más adelante).

Ejemplo

```
XOConnection conn;
XOData Source xgs;
String usuario = "admin"; String usuarioPwd = "admin": // Se crea el Datasource (origen de datos)
xgs = new EXistXOData Source (); // Se establecen las propiedades de conexión xgs. setProperty
("serverName", "localhost"); //nombre del servidor xgs. setProperty ('port", "8080"):://puerto de
conexión. // Se obtiene la conexión conn = xqs.getConnection (usuario, usuarioPwd ) ;
// Se cierra la conexión.
conn. Close () ;
```

En el código se puede ver que la conexión se cierra al final con el método close(). Es obligado no dejar conexiones abiertas que puedan ocasionar problemas de memoria en el código. Además, el código conecta a un localhost, si el sistema gestor estuviese en un servidor remoto, en este parámetro se pondría su dirección.

5. CREACIÓN Y BORRADO DE RECURSOS: CLASES Y MÉTODOS

Como se ha comentado anteriormente, XQJ ofrece un nivel de abstracción más alto que XMLDB y se centra únicamente en ejecutar consultas XQuery obviando la estructura de recursos (documentos) y colecciones que subyace en los sistemas XML nativos, sin dar posibilidad de crear o eliminar colecciones y recursos. Por ello, esta sección se centra en XMLDB y en cómo maneja las colecciones y los documentos, es decir, en cómo se crean y eliminan nuevas colecciones y cómo se añaden y eliminan recursos a esas colecciones.

A modo de ejemplo, la aplicación Aplicacion_Libros, incluida como material digital de este libro, permite añadir y eliminar recursos a colecciones usando XML:DB tal y como se mostrará en esta sección.

5.1. ACCEDIENDO A RECURSOS CON XML:DB

Ya ha quedado claro que los sistemas XML nativos se estructuran en colecciones y dentro de ellas se guardan los documentos que pueden ser XML o no XML. Para el caso de eXist, los documentos no tienen que ser validados

respecto a un esquema XML (XML Schema o DTD) por lo que no tienen que tener estos esquemas asociados y pueden almacenarse sin más.

XMLDB llama a los documentos (XML o noXML) recursos (resources). Es por ello por lo que en este capítulo a los recursos se les puede llamar documentos y a los documentos se les puede llamar recursos, indistintamente.

A continuación se completará el código creado en la Sección 5.4.1 para, partiendo de la colección obtenida, recuperar el recurso (o recursos) guardados en ella. Más concretamente se tratará de recuperar el recurso Libros Inglesxmil que está en la colección dib/Libros/En Ingles (ver Figura 5.1 para recordar la estructura).

En el siguiente código todo el proceso de conexión hasta obtener una colección en coles el mismo que el mostrado anteriormente. Sin embargo, ahora se utiliza la colección colpara recuperar el recurso

y (AQUI IRÍA EL CóDIGO DE LA SECCIÓN 5.4.1}/

Ejemplo

```
Collection col = DatabascManager.get Collection (URI + collection, usuario, usuarioPwd);
nombre recurso-"Libros Ingles.xml";
Resource res=null;
// Se obtiene el recurso "Libros Ingles.xml";
res = (Resource) col. getResource (nombre recurso);
// se usa un molde para convertir res de tipo Resource a XMLResource.
XMLResource xmlres = (XMLResource) res;
// Se sa el contenido del recurso y se muestra en pantalla.
System. out.print ("La salida es: " + Xmlres.getContent () );
```

En este código las clases que se utilizan son: Resource: es una clase contenedor de los datos almacenados en las bases de datos. Sus métodos son: get Content O: recupera el contenido de un recurso.

getId(): recupera el identificador interno y único del recurso o nulo si el recurso es anónimo. Devuelve un java.lang. String.

getParent Collection.0: recupera una instancia de la colección a la que este recurso está asociado. Devuelve por tanto un tipo Collection.

get ResourceType 0: devuelve el tipo del recurso. Devuelve un javalang String indicando si el recurso es un

documento XML (XMLResource) u otro tipo no XML (BinaryResource). set Content0java.lang. Object valor): asocia valor como un contenido para este recurso.

XMLResource. Esta clase extiende a la anterior Resource y proporciona acceso a recursos solo de tipo XML (XMLResource) almacenados en la base de datos. A instancias de esta clase se puede acceder como texto o con API DOM y SAXo Algunos de los métodos destacados de esta clase son:

get ContentAsDOMO y get ContentAsSAXO: estos métodos recuperan elXMLResource como nodo DOM o con un ContentIlandler para SAX, respectivamente.

set Contentas DOM(org.uv.3c.dom. No de content) y set ContentAsSAX0: asigna el contenido del recurso usando un nodo DOM como fuente o usando un ContentHandler SAX, respectivamente.

Recuperar un recurso de la base de datos y manejarlo como un nodo DOM con las funciones que la API de DOM para Java ofrece. ¿Sería posible aplicar una plantilla XSLT a ese DOM recuperado? ¿Cómo?

5.2. CREANDO RECURSOS CON XML:DB

En una colección (objeto Collection) se pueden añadir nuevos recursos XML y no XML. Para ello se necesitan las siguientes clases y métodos:

Collection: esta clase representa una colección de recursos (Resources) almacenada en la base de datos XML. Los métodos más relevantes de esta clase para añadir nuevos recursos son:

store Resource (Resource res): almacena en la colección un recursores proporcionado por parámetro. removeResource (Resource res): elimina de la colección un recursores que se le pasa por parámetro.

- Otros métodos interesantes de esta colección, útiles para crear y eliminar nuevos recursos, son list Resources0, que devuelve un array de String con todos los ids de los recursos que tiene la colección;get ResourceCount0 obtiene el número de recursos almacenado en la colección; y createResource(java.lang Stringid.java.lang. String type), que crea en la colección un nuevo recurso vacío con id y tipo pasados por parámetro.

El siguiente código muestra una función que tiene como parámetro contexto, que es la colección (Collection) donde se almacenará el recurso, y archivo que es un File que representa el archivo que se quiere añadir a la colección. Como se puede apreciar, este código tiene un punto más de complejidad ya que incluye excepciones para la captura de errores.

Ejemplo

```
public void asignar RecursoBD (Collection contexto, Fila archivo) throws ExcepcionGestorBD. try {
//Crea un recurso vacío
Resource nuevo Recurso = contexto. CreateResource (archivo.getMame (), "XMLResource");
//Le asigna el contenido del archivo al nuevo recurso vacío nuevo Recurso. set Content (archivo);
//almacena el recurso en la colección. contexto. StoreResource (nuevo Recurso) ;
} catch (XMLDBException e) {
throw new ExcepcionGestorBD ( "error XMLDB : " + e.getMessage () );
```

El código crea un recurso vacío usando como identificador (id) el nombre del archivo (podría ser c:\\Libros Favoritos2.xml) y le indica que es de tipo XMLResource (recurso XML). A ese nuevo recurso vacío creado le añade el contenido del archivo (podría ser el contenido de c:\Libros Favoritos2.xml). Hecho esto, almacena el recurso en la colección (que podría ser /db/Libros/En Ingles).

5.3. BORRANDO RECURSOS CON XML:DB

Si se ha entendido el proceso de creación de nuevos recursos, el proceso de borrado se puede definir como mucho más sencillo. Aquí vuelve a intervenir la clase Collection. Los métodos más destacados para el borrado de recursos son:

removePesource (Resource res), que elimina el recursores de la colección. get Resource(java.lang Stringid), que recupera un recurso a través de su id.

Actividad

2) Implementar una pequeña aplicación que añada un recurso a una colección de una base de datos, por ejemplo,

como la de la Figura 5.1. La aplicación debe tener: a. Un botón "Añadir Recurso" que permita seleccionar, con una caja de diálogo, un documento XMLalmacenado

en el disco duro y guardario en una colección de una base de datos exist. b. Un botón "Eliminar Recurso" que permita eliminar un recurso dando el nombre del id con el que se almacenó. D). Utilizar eXIst Client Shell para comprobar los cambios en la base de datos.

6. CREACIÓN Y BORRADO DE COLECCIONES: CLASES Y MÉTODOS

Una vez visto en la Sección 5.5 el acceso, creación y borrado de recursos en colecciones, en esta sección se tratará la creación y borrado de las propias colecciones. Al igual que ocurre con los recursos, solo XML-DB permite una gestión directa con las colecciones.

A modo de ejemplo, la aplicación Aplicacion Libros, incluida como material digital de este libro, permite añadir y eliminar colecciones a una estructura de árbol usando XMLDB tal y como se mostrará en esta sección.

6.1. CREACIÓN DE COLECCIONES CON XML:DB

Evidentemente, para la creación de colecciones una de las clases que participarán en esta gestión será Collection. Sin embargo no es la única, también es necesaria CollectionManagementService que es una clase que extiende Service y que permite la gestión básica de colecciones en las bases de datos. Los métodos necesarios para crear colecciones son:

De la clase Collection es necesario el método get Servicejava.lang String name, javalang String version) el cual devuelve una instancia de un servicio según los parámetros nombre y versión. Si no es posible crear un servicio con esos parámetros devolverá null.

De la clase CollectionManagementService es necesario el método createCollection (java. lang String name), que crea una colección en la base de datos con el nombre name pasado como parámetro.

El siguiente código muestra una función que tiene como parámetro contexto, que es la colección (Collection), donde se creará una colección hija, y NevColec que será el nombre de la colección. Como se puede apreciar, en este código también se incluyen excepciones para la captura de errores.

Ejemplo

```
public Collection anadir Coleccion (Collection contexto, String new Colec) throws ExcepcionGastorBD.

Collection newCollection=null;

try { // Se crea un Nuevo servicio desde el contexto. CollectionManagementService mgtService -

( CollectionManagementService) contexto.getService ( "CollectionManagementService", ; ("1.0' // Se crea una nueva collection con el nombre new Colec codificado //UTF8. La colección nueva se devuelve en new Collection (Collection) new Collection = mgtService. create Collection (new String (UTF8. encode (new Collec)));

} catch (XMLDBException e) { throw new ExcepcionGestorBD ( "Error añadiendo colección: " + e. getMessage () ) ;

return new Collection;
```

Actividad

3.1 Instalar en local el Sistema gestor native eXist:

a) Dentro del asistente de instalación definer un usuario "admin" y una clave también "admin".

b) Instalar el servicio y comprobar que se está ejecutando (escuchando) en el administrador de servicios (procesos).

c) Ejecutar el cliente icluido en la instalación llamado eXsit Client Shell. Relizar una tarea de investigación para familiarizarse con el entorno. ¿Como se crean nuevas colecciones? ¿Como se añaden recursos a las colecciones?

d) Crear una estructura similar a la ofrecida previamente. Para ello se pueden utilizer los documentos XML ofrecidos como material digital de este libro.

e) Ejecuta una consulta XPath en el entorno para comprobar cómo devuelve el resultado.

3.2 Recuperar un recurso de la base de datos y manejarlo como un nodo DOM con las funciones que la API de DOM para ÇJava ofrece. ¿Sería possible aplicar una plantilla XSLT a ese DOM recuperado? ¿Cómo?

3.3 Implementar una pequeña aplicación que añada un recurso a una colección de la base de datos. La aplicación debe tener:

3.3.1 Un botón añadir recurso que permita seleccionar, con una caja de diálogo, un documento XML almacenado en el disco duro y guardarlo en una colección de una base de datos eXsit.

3.3.2Un botón eliminar recurso que permita elminar un recurso dando el nombre del id con el que se almacenó.

3.4 Utilizar eXsit Client Shell para comprobar los cambios en la base de datos.

3.5 Partiendo de la aplicación creada en Actividad 3.4, hay que añadir nueva funcionalidad que permita crear nuevas colecciones y borrarlas. La nueva funcionalidad debe ser:

a. Un botón "Añadir Colección" que permita escribir en un diálogo el nombre de la colección padre y el de la nueva colección hija que se creará, y luego crear la colección.

b. Un botón "Eliminar Colección" que permita eliminar una colección introduciendo en un diálogo el nombre de la colección padre y el de la colección hija que se eliminará.

c. ¿Es posible eliminar directamente colecciones que tienen recursos o colecciones hijas asociadas?

3.6 Partiendo de la aplicación creada en Actividad 3.4 añadir nueva funcionalidad que permita ejecutar consultas XQuery Update Extension. La nueva funcionalidad debe tener:

a. Una caja de texto en la que escribir la consulta XQuery Update Extension (se puede empezar con las vistas en la Sección 3.7.1).

Actividad

b. Incluir un botón que ejecute la consulta y una caja de texto o etiqueta en la que informar de si hay algún error o todo el proceso ha sido correcto.

3.7 Partiendo de la aplicación creada en la Actividad 3.4, añadir nueva funcionalidad que permita ejecutar consultas XUpdate. La nueva funcionalidad debe tener:

a. Una caja de texto en la que escribir la consulta XUpdate (se puede empezar con las vistas en la Sección 3.7.3).

b. Incluir un botón que ejecute la consulta y una caja de texto o etiqueta en la que informar de si hay algún error o todo el proceso ha sido correcto.

3.8 Partiendo de la aplicación creada en Actividad3.4, añadir nueva funcionalidad que permita ejecutar consultas XQuery utilizando XQJ, XML:DB o ambas tecnologías. La nueva funcionalidad debe tener:

a. Una caja de texto en la que escribir la consulta XQuery (se puede practicar con las vistas en la Sección 5.8.1).

b. Incluir un botón que ejecute la consulta y una caja de texto o etiqueta en la que informar de si hay algún error o de si todo el proceso ha sido correcto.

Índice