Unidad 7

Bases de datos de XML

Acceso a datos

Índice

Introducción

Las bases de datos XML se usan cada vez menos a favor de las bases de datos relacionales, las cuales, en general, cuentan con mejores características, pero no por ello se han dejado de usar, ya que sus características únicas las hacen ideales para el almacenaje de información con ciertas necesidades.

Con esto en mente estudiaremos los distintos tipos de bases de datos XML existentes junto con los SGBD y API apropiados para trabajar con ellas en función de nuestras necesidades.

Veremos también algunas de las sentencias más comunes y útiles que podemos emplear en eXist con cada una de las API, como puede ser la creación o eliminación documentos o la modificación de contenidos.

Al finalizar esta unidad

- Conoceremos los distintos modelos de bases de datos XML que existen, así como los distintos sus características y elementos principales, de modo que seamos capaces de discernir claramente las situaciones óptimas para emplearlos.
- Habremos estudiado los diferentes gestores, los SGBD, tanto libres como comerciales y su función.
- Sabremos las diferentes API que podemos emplear, XML:DB, XQuery y XQJ, así como sus propiedades y características junto algunas de sus sentencias más importantes, como pueden ser incluir o eliminar documentos o modificar la información de estos.

7.1. XML como soporte para almacenamiento e intercambio de datos

El éxito del lenguaje XML se debe a diversos factores como son su estandarización por W3C, su estructura jerárquica y sencilla, y la capacidad de contener cualquier tipo de dato. Debido a este desarrollo de XML han surgido numerosos elementos que trabajan con XML ya sea empleándolo o modificarlo, en este tema nos centraremos en el empleo de XML como soporte de almacenamiento de datos y sus aplicaciones y características.

- **XML centrado en documentos (document-centric XML).** Empleo de documentos XML de la misma forma que los anteriores HTML, siendo construidos por personas, en un lenguaje natural y sin una estructura fija y regulada, con el fin de emplearse en procesos manuales.
- **XML centrado en datos (data-centric XML).** Empleo de XML con una estructura fija y regulada para almacenamiento de datos. Generalmente se crean automáticamente y se destinan a programas como XML Schema, XPath, XSL...

Estos dos pasos son extremos, generalmente se emplean puntos intermedios con características de ambos.

7.2. Alternativas para el almacenamiento de documentos de XML

La conversión en datos persistentes de la información recogida en los documentos XML puede realizarse desde distintas maneras en función a diversos factores, como el tipo de documento o el uso previsto de dicha información.

- **Sistemas de ficheros.** Se emplea una jerarquía de directorios para almacenar los distintos ficheros XML que conservan la información, no posee más sistemas de búsqueda con excepción de los sistemas operativos. Cualquier tipo de búsqueda o modificación debe hacerse manualmente.
- **Bases de datos con capacidades para XML (XML-enabled).** Existen distintos tipos de esta, ya que se basa en el empleo de un sistema de traducción para documentos XML, el cual nunca será tan eficiente como las bases de datos destinadas a XML, lo que provoca un menor rendimiento y consumo de espacio de almacenamiento.
- **Bases de datos nativas de XML.** Bases de datos destinadas a XML y optimizadas para ello, poseen diversos elementos con los que trabajar, ya sean lenguajes de consulta, o modificación, XPath, XSL, etc. Destacan las API que incluyen, como XQJ o XML:DB.

7.3. Almacenamiento de XML en SGBD relacionales

En las bases de datos relacionales podemos encontrar distintas formas para el almacenamiento persistente de datos en documentos XML, aunque no es óptimo el empleo de XML con bases relacionales es utilizado por:

- El empleo de las bases de datos relacionales se debe principalmente al uso extendido de estas, por lo que salvo que el empleo de una base XML nativa conlleve una ventaja significativa, o algún otro factor externo, no se suele emplear.
- Permite el almacenamiento de documentos XML completos y su recuperación, sin importar su extensión, mediante el empleo de columnas CLOB (carácter large object), BLOB (bynary large object) o, solo para documentos pequeños, en las columnas tipo VARCHAR.
- Permite, para los documentos XML *data-centric*, extraer los datos que contienen para su almacenamiento y volver a reformarlos a la hora de su empleo en un proceso automático.
- SQL en su variante SQL/XML incluye:
  - Los tipos de datos para el almacenamiento de los documentos XML.
  - Correspondencias entre SQL y XML que faciliten la interacción.
  - Capacidad de emplear XQuery para consultas SQL lo que permite la combinación y relación con documentos XML.
  - Creación de documentos de XML a partir de los resultados de consultas de SQL.
- Existencia de las bases de datos relacionales con XML-*enabled*, mediante SQL/XML.
- Gran capacidad de transacción de las bases de datos relacionales a diferencia de las bases de datos nativas.

Debido al empleo de XML en las bases relacionales la diferencia entre ambas es cada vez menor, por ejemplo, con la adquisición por parte de SQL de muchas de las características de XML, o la inclusión de soporte para tecnologías auxiliares de XML como XSL. En la actualidad destaca XML DB ya que posee muchas de las características y tecnologías de XML a la vez que posee un soporte para SQL/XML.

7.4. Características de las bases de datos de XML nativas

Este tipo de base de datos XML nativa permiten un gran acceso a los documentos XML y su contenido, sin importar su tipo o circunstancias, pero por otro lado provoca que, aunque pueda almacenarlos, los tipos de documentos con los que puede trabajar. Así como las acciones que puede realizar sobre estos, son realmente limitadas.

Las bases de datos XML nativas suelen poseer las siguientes características:

- **Colecciones.** Incorporación de un SGBD, sistema gestor de bases de datos, de XML que permite la organización de las colecciones en función a la naturaleza de este.
- **Validación.** Empleo de DTD o esquemas XML para la validación de los documentos XML almacenados.
- **Mecanismos estándares para consulta y transformación de documentos de XML.** Compatibilidad con XPath, XQuery y XSL con el estándar de W3C.
- **Mecanismos estándares para modificación de documentos.** Empleo de lenguajes de modificación de documentos como XQuery.
- **Soporte para API estándares.** Los API más importantes y utilizados son XQJ y XML:DB, aunque, generalmente, cada base de datos poseerá su propia API en la que las API estándar se apoyarán.
- **Indexación.** La indexación es fundamental para una consulta asequible de los contenidos, pero la organización de las bases XML nativas dificulta de gran manera este proceso, ya que permite documentos de diferente tipología de contenido juntos. Con el fin de crear una indexación en estos casos es necesario que se flexibilicen los parámetros de búsqueda de modo que seamos capaces de localizar el documento deseado, resultando finalmente en índices poco estandarizadas. Otro de los inconvenientes es que todas las actualizaciones, ya sean de ficheros completos o datos de estos, se deben indexar manualmente ya que no existe ningún tipo de automatización en este proceso.
- **Transacciones.** Las XML nativas no están diseñadas para soportar transacciones, por lo que son realmente ineficientes y con considerables problemas, aunque estos se pueden paliar con XQJ o con SGBD desde API. A pesar de estos añadidos las XML nativas nunca serán recomendables por sus buenas transacciones.

7.5. Gestores comerciales y libres

Aunque están perdiendo terreno desde hace un tiempo frente a las bases de datos relacionales, las bases de datos XML nativas se siguen empleando y existen una multitud de SGBD para emplearlas fácilmente. Tanto libres como comerciales.

Algunos de estos SGBD son:

- **Tamino.** SGBD comercial con un buen rendimiento, pero precio igualmente alto, destaca por su diseño optimizado para XML y su indexación.
- **Marklogic.** SGBD comercial con naturaleza de base de datos multimodelo con una vista unificada de los datos con indiferencia del almacenamiento empleado. Crea una organización jerárquica con colecciones con, al menos, un elemento común, pudiendo estar el mismo documento en diversas colecciones. Existe una versión gratuita para uso no comercial.
- **eXist.** SGBD libre, destaca por la jerarquía de sus colecciones, asignando un identificador único para cada uno de los documentos, manteniendo automáticamente índices para ellos.
- **BaseX.** SGBD libre que puede contener documentos XML y *raw*. Destaca por no crear o borrar explícitamente colecciones, sino implícitamente en función de la existencia de *paths* específicos.
- **Sedna.** SGBD libre en C/C++, en lugar de Java, muy eficiente y con un API basado en un protocolo binario en lugar de los habituales. Por desgracia se dejo de actualizar a finales de 2012.

7.6. Instalación y configuración del SGBD de XML nativo eXist

Como la mayoría de SGBD eXist emplea Java por lo que puede emplearse en cualquier sistema operativo que lo use, como Windows o Linux.

Para poder instalar este SGBD se deben seguir los siguientes pasos:

En primer lugar, se debe ejecutar el programa de instalación, asignando una contraseña de "admin" si es necesario. Tras realizar este proceso podremos encontrar las opciones de eXist en el menú de inicio, donde podremos lanzarlo o instalarlo a nuestro gusto, si se va a trabajar con el se recomienda la instalación, ya que esta nos dará una mayor seguridad frente a perdidas de información por cortes repentinos.

Una vez iniciado eXist podemos acceder a su *dashboard*, panel de control, desde el servidor web en la dirección https://localhost:puerto. Dentro de este podemos encontrar las siguientes opciones:

- **Usermanager.** Permite crear nuevos usuarios y seleccionar sus propiedades. En el caso de que la contraseña "admin" no se hubiera asignado se realizaría en este momento.
- **Collections.** Permite crear, eliminar y gestionar las colecciones. La instalación gestiona una única base de datos que forma su jerarquía al definir unas colecciones sobre otras. Estas colecciones pueden contener tanto documentos XML como de otros tipos.
- **eXide.** Un IDE para XQuery que permite entre otras cosas visualizar búsquedas con este y editar los documentos XML con un editor de texto.
- **Java Admin Client.** Permite realizar consultas y diversas tareas de gestión como:
  - Importar ficheros o directorios desde un sistema de ficheros.
  - Gestionar colecciones.
  - Realizar diversas acciones con documentos, crear, borrar, mover y renombrar.
  - Reindexar colecciones.
  - Gestionar los usuarios y los permisos que poseen.
  - Realizar o recuperar copias de seguridad.
- **Package Manager.** Permite instalar paquetes y utilidades adicionales a nuestra discreción, como puede ser el recomendable "eXist-db Demo Apps".
- **Shutdown.** Permite parar y cerrar eXist, es necesario si no se llega a instalar, ya que en caso de no emplearlo es posible que se pierdan los datos no guardados.

7.7. API para gestión de bases de datos nativas de XML

Las bases de datos nativas XML suelen incluir sus propias API, aunque suelen emplear, o coordinarse con las API estándar XML:DB y XQJ.

- **XML:DB.** La primera API estándar para XML creada, tuvo su última revisión en 2001.
- **XQJ, XQuery for Java.** API de 2009 creado por JCP, Java Community Process, no se incluye en la biblioteca estándar de clases Java, por lo que es necesario descargarlo por separado.

Como API XQJ se basa en XQuery y, por lo tanto, trabaja exclusivamente con XQuery y por tanto XML. Por otro lado, XML:DB, además de poder emplear XQuery, permite otras muchas funciones de creación y borrado de documentos u operaciones de gestión.

7.8. La API XML:DB

A pesar de su antigüedad y falta de actualización desde 2001, esta API sigue siendo ampliamente empleada por muchos SGBD de XML nativos, como eXist, debido a las grandes prestaciones que ofrece y a la gran cantidad de operaciones que permite realizar sobre las bases de datos, lo que otorga gran libertad a los desarrolladores, especialmente si se elige la base de datos adecuada a nuestras necesidades.

Algunas de las acciones que esta API puede realizar, a diferencia de XQJ, son operaciones realizadas sobre colecciones y operaciones de creación y eliminación de documentos de XML que se encuentran dentro de las colecciones.

Es necesario recordar que para el empleo de eXist es necesario añadir algunos ficheros al proyecto que vayamos a iniciar, entre los que se encuentran los ficheros exist.jar, un fichero que inicie por xmldb-db-api y los que se encuentren en el directorio lib/core.

En el caso de que no poseamos alguno de los ficheros que necesitemos se lanzará la excepción ClassNotFoundException. En este caso será necesario incluir el fichero faltante, si nos cuesta localizarlo podemos emplear buscadores como www.findjar.com.

7.8.1. Establecimiento de conexiones y acceso a servicios con XML:DB

Para implementar XML:DB es necesario que se realicen dos pasos previos: crear una clase para la interfaz Database y al mismo tiempo aportar a la base de datos una cadena de conexión, con el formato adecuado para el SGBD empleado.

XML.DB para distintas bases de datos XML

SGBD	Clase implementadora de Database	Cadena de conexión
eXist	org.exist.xmldb.DatabaseImpl	Xmldb:exist://host:puerto/exist/xmlrpc/db/colección
BaseX	org.basex.api.xmldb.BXDatabase	Xmldb:basex://localhost:puerto/basedatos
Sedna	net.cfoster.sedna.DatabaseImpl	Xmldb:sedna://host:puerto/basedatos/colección

Es recomendable saber que eXist no realiza la conexión hasta que esta se solicita al llevar a cabo una operación, por lo que no se detectarán errores en ella hasta que se intente, llegando a ignorar, si es localhost, el puerto indicado y empleando el configurado por eXist.

Existen otros servicios, como los proporcionados por Sedna, que eXist no posee, como:

- TransactionService.
- SednaUpdateService.
- XQueryService.
- IndexManagementService.
- ModuleManagement-Service.

```
…
public class XMLDB_Conexion {
private static Collection obtenC0leccion ( String
nomC0lec) throws Exception {
Database dbDriver;
Collection colec;
dbDriver = (Database )
CIass.forName(“.org.exist.xmldb.DatabaseImpl”).
newInstance( );
DatabaseManager.registerDatabase(dbDriver);
colec = DatabaseManager.getC0llection(
“xmldb:exist://localhost:7650/exist/xmlrpc/db” +
nomC0lec,
“(usuario)”, “(contraseña)” );
return colec;
}
…
}
```

7.8.2. Creación y borrado de colecciones y documentos con XML:DB

Podemos crear o borrar colecciones mediante el empleo de los métodos `createCollection` y `removeCollection`.

```
…
cmServ.createCollection(“prueba”)
cmServ.removeCollection(“prueba”)
…
```

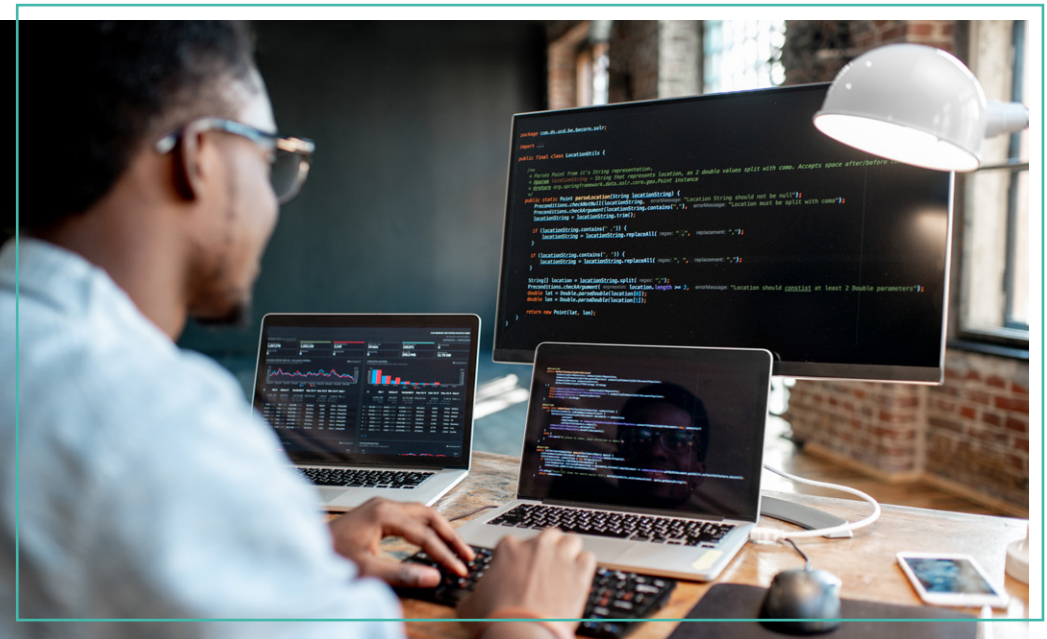
Para la creación y eliminación de documentos emplearemos el método `createResource`, y, adicionalmente, lo almacenaremos con `storeResource`.

Una vez creado el documento podremos modificar su contenido mediante la interfaz `XMLResource` empleando los métodos `setContent`, `getContent`, `setContentAsDOM`, `getContentAsDOM`, `setContentAsSAX`, `getContentAsSAX`.

En el caso de que se desee, en eXist, almacenar documentos no XML, los cuales eXist denomina binarios, deben emplear `BinaryResource` en lugar de `XMLResource`.

Para la eliminación de documentos emplearemos el método de `removeResource`.

```
…
Ejemplo = (XMLResource) col.createResource(“Usuarios.xml”,
XMLResource.RESOURCE TYPE);
Ejemplo.setContent(“<usuarios>\n”
+ “<usuario DNI=\ “75954585S\”>”);
…
Colec.removeResource(col.getResource(“Usuarios.xml”));
…
```



7.9. El lenguaje XQuery

XQuery, al igual que XPath, nació como un lenguaje de consulta con estándar W3C destinado a trabajar con XML. Podemos emplear todas las consultas de XPath 3.1 en XQuery 3.1. Con el tiempo este lenguaje de consulta, debido a la necesidad, pasó a ser un lenguaje para operaciones de modificación de XML.

Dado que XML puede contener "lenguaje natural", hecho por personas para ser leído por personas, XQuery incluye una extensión para este tipo de texto denominado XQuery Full Text.

7.9.1. Consultas con XQuery

Una característica de XQuery es que es posible realizar tanto una consulta sobre un documento como una consulta sobre una colección.

Las sentencias de XQuery emplean un sistema FLWOR semejante al empleado por SQL.

- **XQuery:** *for, let, where, order by, return.*
- **SQL:** *select, from, where, group by, having, order by.*

Cláusulas de las sentencias FLWOR de XQuery

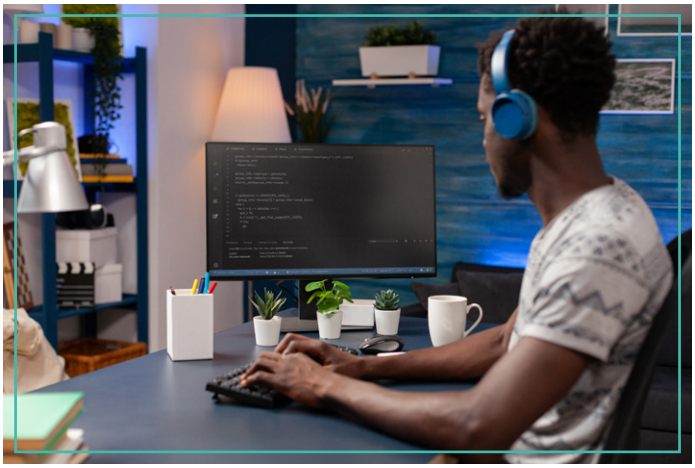
Sentencia	Descripción
`FOR`	Vincula variables a expresiones en XPath, estas pueden ser varias y se guardarán durante el resto del proceso.
`LET`	Asigna a una variable el resultado de una evaluación de una expresión en XPath, para evitar repetirla, si existe más de un resultado los concatenará.
`WHERE`	Condiciones para filtrar los resultados de `FOR` y `LET`.
`ORDER BY`	Ordena los resultados de `FOR` y `LET`.
`RETURN`	Evalúa una expresión con sus resultados anteriores. Se puede añadir entre llaves `{}` sentencias XQuery.

Ejemplos de FLWOR	
Ejemplos	**Explicación**
`for $n in doc('/db/ejemplos/Usuarios.xml')` `return $n/Usuarios/usuario`	Devuelve los datos de todos los usuarios como elementos nombrados *usuario* del documento.
`for $n in doc('/db/ejemplos/Usuarios.xml')` `return $n/Usuarios/usuario/Apellidos/text()`	Permite que en lugar del elemento anterior solo se devuelva el texto, de todos los usuarios de la columna deseada, en este caso, *Apellidos*.

`for $n in doc('/db/ejemplos/Usuarios.xml')`
`/Usuarios/usuario`
`where substring($n/CP,1,2)="30"`
`return concat($n/apellidos,"-", $n/string(@DNI))`
	Empleamos where para acotar la búsqueda por CP, devolviendo como resultado la concatenación de Apellido – DNI de cada uno de los usuarios que cumplen con la condición where.

`for $n in doc('/db/ejemplos/Usuarios.xml')`
`/Usuarios/usuario`
`order by number ($n/CP)`
`return`
`<cli dni="{&n/string(@DNI)}" nom="{&n/nombre}"></cli>`
	Devuelve, ordenados numéricamente por CP, el DNI y nombre de los usuarios.

`let $cl:=doc('/db/ejemplos/Usuarios.xml')`
`/Usuarios/usuario`
`return<usuariosvip>{$cl}</usuariosvip>`
	Devuelve resultados condicionados por una asociación de una variable.



7.9.2. Sentencias de modificación de datos con XQuery

eXist también posee la llamada *XQuery Update Extension*, la cual contiene numerosas sentencias de XQuery que permiten la modificación de los datos en XML., como son:

- **into**: introducción de datos.
- **following**: introducción de datos tras el elemento especificado. También existe preceding.
- **with**: cambia los valores de un elemento.

Ejemplo de sentencias de modificación de XQuery	
Ejemplo	**Explicación**

`Update insert`
`<usuario DNI="76534825F">`
`<fechareg>26-03-2022</fechareg>`
`</usuario>`
`Into doc('/db/ejemplos/Usuarios.xml')/usuarios`
	Inserta los datos de un nuevo cliente, este se insertará como el último elemento de usuarios.

`Update insert`
`<usuario DNI="76534825F">`
`<fechareg>26-03-2022</fechareg>`
`</usuario>`
`following doc('/db/ejemplos/Usuarios.xml')/usuarios`
`/usuario[@DNI="45685215D"]`
	Inserta los datos de un nuevo cliente, este se insertará tras el elemento designado.

`Update value doc('/db/ejemplos/Usuarios.xml')/usuarios`
`/usuario[@DNI="45685215D"]/primerape`
`With "García"`
	Sustituye el valor del elemento designado con el valor dado.

`Update value doc('/db/ejemplos/Usuarios.xml')/usuarios`
`/usuario[@DNI="45685215D"]`
`With`
`<usuario[@DNI="45685215D">`
`<primerape>García</primerape>`
`</usuario>`
	Sustituye el elemento designado completo, no solo el valor, con el dado.

7.10. La API XQJ

XQJ es una API que permite ejecutar sentencias a XQuery, y sus extensiones, desde programas Java, de la misma manera que JDBC hace para SQL.

XQJ no es una de las clases estándar de Java que aparecen en su biblioteca. Podemos encontrar los Javadocs de XQJ en la siguiente página: http://xqj.net/javadoc/

7.10.1. Establecimiento de conexiones con XQJ

Para poder trabajar con XQJ es necesario preparar ciertos elementos:

- **Incluir en el proyecto el archivo Xqjapi.jar.** Se debe incluir el fichero, ya que es este el que contiene la API XQJ.
- **Establecer conexión.** Crearemos una instancia de la clase con la interfaz XQDataSource, tras esto se proporcionan los valores para los parámetros con setProperty. Podemos encontrar los parámetros de conexión en el *driver*, en Compliance Definition Statement, no todos los parámetros requieren ser rellenadas.
- **Realizar una operación para comprobar el funcionamiento de la conexión.** La conexión solo se realiza durante las operaciones, por lo que para comprobar su funcionalidad y posibles errores es necesario implementar una operación y observar si la conexión se realiza correctamente.

Cadenas de conexión para los drivers de XQJ para bases de datos nativas		
SGBD	Clase implementada XQDataSource	Propiedades para conexión
MarkLogic	net.xqj.marklogic.MarkLogicXQDataSource	serverName, databaseName, port, user, password, description y mode
eXist	net.xqj.exist.eXistXQDataSource	serverName, port, user, password y description
BaseX	net.xqj.basex.BaseXXQDataSource	serverName, databaseName, port, user, password y description
Sedna	net.xqj.sedna.SednaXQDataSource	serverName, databaseName, port, user, password y description

Como ya se ha indicado no todas las propiedades requieren ser completadas, como, por ejemplo, la descripción.

7.10.2. Consultas con XQJ

La realización de consultas con XQJ guarda gran relación con JDBC. Debemos establecer la conexión `XQConnection` para obtener con esta la expresión `XQExpression`, una vez obtenida podemos ejecutar sobre ella la consulta `executeQuery(Strig query)` con lo que obtendremos un resultado, semejante a `ResultSet`, mediante `XQResultSequence`. Es posible realizar iteración empleando `next()`.

Podemos obtener el resultado como `XMLStreamReader` o como `Node` de DOM, empleando, respectivamente, `getItemAsStream()` y `getnode()`.

```
…
private static XQConnection obtenConexion( ) throws
ClassNotFoundException,
InstantiationException, IllegalAccessException,
XQException {…}
private static void muestraErrorXQuery (XQException e )
{…}
public static void main (String[] args) {
XQConnection c = null;
try {
c = obtenConexion();
String cad = “doc(‘/db/Ejemplos/Usuarios.xml’)/usua-
rios/usuario/
Nombre”;
XQExpression xqe = c.createExpression();
XQResuItSequence xqrs = xqe.executeQuery (cad);
while (xqrs.next()) {
XMLStreamReader xsr = xqrs.getItemASStream();
…}
…
```

7.10.3. Modificaciones de documentos con XQJ

Las modificaciones se realizan empleando `XQExpression` con el método `executeCommand`.

```
…
XQExpression xqe = c.createExpression();
Xqe.executecommand(cad);
…
```

7.10.4. Transacciones con XQJ

A diferencia de eXist, XQJ si permite la creación de transacciones, conjunto de operaciones llevadas a cabo como una que siguen los requisitos ACID, si el SGBD lo permite, de una manera similar a JDBC.

Al tratarse de un conjunto de operaciones cabe la posibilidad de que alguna de ellas produzca errores y que el resto funcione, lo que puede causar problemas si no se lleva a cabo correctamente.

Al completar una transacción se realiza una confirmación automática con la base de datos, si se desea eliminar esta función es necesario emplear `setAutoCommit(false)`, si se emplea la revisión solo se lleva a cabo al utilizar `commit()`.

Podemos llevar a cabo una rectificación para eliminar todos los cambios realizados por la transacción con `rollback()`. En case de error debemos capturar la excepción e implementar `rollback`.

www.universae.com