6 Almacenamiento de información

OBJETIVOS DEL CAPÍTULO

- ✓ Buscar a través de Internet las necesidades de almacenamiento que tienen las grandes empresas tecnológicas.
- ✓ Descubrir, buscando qué es el método Fermi, cómo estimar cualquier necesidad de almacenamiento (sin datos suficientes).
- ✓ Conocer los tipos de bases de datos existentes.
- ✓ Descubrir cómo unificar un conjunto de bases de datos y administrarlos a través de los Sistemas de Gestión de Bases de Datos (SGBD).
- ✓ Mostrar la importancia de las transacciones y conocer que características deben implementar los SGBD para conseguirlas (ACID).
- ✓ Buscar y comparar distintos SGBD.
- ✓ Conocer qué es SQL y cómo XQuery permite realizar las mismas funciones pero en BD XML nativas.
- ✓ Convertir una BD relacional a un conjunto de documentos XML equivalentes.
- ✓ Utilización de Qizx Studio como motor de BD XML nativo.
- ✓ Utilizar expresiones FLWOR en XQuery para realizar consultas, inserciones, modificaciones y borrados de la BD. También para generar documentos XML o XHTML.
- ✓ Exportación de BD a ficheros XML y la importación en la BD de documentos XML.
- ✓ Conocer otras librerías y APIs que permiten tratar los documentos XML a través de lenguajes de programación como Java o C/C++.

Eric Schmidt, CEO de Google hasta enero del 2011, realizó la siguiente afirmación en la conferencia de Lake Tahoe en California (EE.UU.):

"Cada dos días creamos tanta información como la que se generó desde los albores de la humanidad hasta el 2003".

Figura 6.1. *Eric Schmidt, antiguo CEO de Google*

Para obtener más información, vea las páginas *http://www.isidoroporquicho.com/eric-schmidt-every-2-days-we-create-as-much-i* y *http://techcrunch.com/2010/08/04/schmidt-data/*.

Aún cometiendo algún error numérico en la estimación de Schmidt, parece claro que de manera constante y diaria producimos ingentes cantidades de información que debemos almacenar y procesar. Esta afirmación es tremendamente reveladora en la era de la información en la que vivimos, pero:

- ¿Cómo es posible que podamos generar toda esa información en tan poco tiempo?
- ¿Cómo es posible que además podamos almacenarla?

Un ejemplo de generación masiva de información puede ser el acelerador de partículas llamado Gran Colisionador de Hadrones. Éste se encuentra ubicado en la Organización Europea para la Investigación Nuclear (CERN) de Ginebra-Suiza. La red Física-Matemática-Informática que se ha desplegado permitirá capturar características, velocidades y trayectorias de las partículas subatómicas que se conocen e incluso descubrir nuevas partículas. Para ello han desarrollado un sistema llamado *Grid-Computing* en el que se manejarán la enorme cantidad de datos producidos por el acelerador. Estiman que generarán en torno a 40 Terabytes de datos por día, enviándose copias de los datos a todas las instituciones académicas de índole mundial que trabajan en el proyecto. Llegan a afirmar que ellos solos podrán generar 15 petabytes al año (y el tamaño sigue creciendo).

Sin duda las novedades que se presentan en los congresos tecnológicos sobre las tecnologías de la información, almacenamiento y procesamiento hacen pensar que no solo podemos generar y almacenar la información, sino también tratarla y realizar operaciones con ella. Pero debe existir alguna manera de seleccionar, catalogar, ordenar la información relevante y desechar aquella que esté duplicada o sea superflua, sin que el rendimiento en el acceso penalice los tiempos. Ese sistema se llama Sistema de Gestión de Bases de Datos (SGBD).

¿SABÍAS QUE...?

El CERN fue fundado en 1954 y está financiado por 20 estados miembros de la Unión Europea. Para el período 2011-2015 tiene un presupuesto de unos 3800 millones de euros. La mayor contribución del CERN a la humanidad fue la creación del protocolo *World Wide Web* (www), creada por Tim Berners-Lee para acceder a la información del centro a través de los sistemas de comunicaciones internos.

¿SABÍAS QUE...?

Google y Microsoft utilizan, entre otras muchas, una técnica de selección de personal llamada "Cuestiones de Fermi". Con esta técnica se permite medir las capacidades de abstracción de los aspirantes ante problemas complejos por la insuficiencia de información suficiente.

ACTIVIDADES 6.1

- Un Kilobit son 2^{10} bits. Un Megabit son 2^{20} bits. Un Gigabit son 2^{30} bits. ¿Qué nombres tienen las siguientes capacidades?

  a. 2^{40} bits
  b. 2^{50} bits
  c. 2^{60} bits

Para obtener más información, vea la página *http://es.wikipedia.org/wiki/Petabyte*.

- Buscar por Internet cuanta información está actualmente almacenada en los servidores de Google. ¿Y en los servidores de Facebook/Tuenti?
- ¿Qué es un problema de Fermi? Buscar por Internet dos ejemplos en los que se utilice esta metodología.

Para obtener más información, vea la página *http://es.wikipedia.org/wiki/Problema_de_Fermi*.

- ¿Cómo puede ayudar una estimación de Fermi para almacenar datos en un SGBD?
- Estimar, con el método de Fermi, qué tamaño de base de datos deberíamos tener para almacenar en imágenes toda la vida de una persona en España.

6.1 SISTEMAS DE ALMACENAMIENTO DE LA INFORMACIÓN

Una Base de Datos (BD) es una herramienta que permite organizar los datos que tienen algún tipo de relación entre sí, de manera que son almacenados y utilizados a través de un interfaz de comunicación que facilita su gestión dentro de una entidad u organización. Existen muchas maneras de clasificar los distintos tipos de bases de datos pero podemos resumirlo según la variabilidad de los datos que se almacenen en ellas:

- **Datos estáticos**: Datos que nunca cambiaran en el tiempo (solo lectura).
- **Datos dinámicos**: Datos que pueden modificarse de cualquier manera, a lo largo del tiempo.

Salvo contadas excepciones (datos históricos, estadísticas, análisis clínicos que nunca cambiarán desde que se tomaron), las bases de datos que se utilizan hoy en día son dinámicas. Las bases de datos estáticas suelen utilizarse cuando se requiere un alto rendimiento en las consultas, quedando muy penalizadas si tuviera que actualizarse algún dato posteriormente.

Los datos que se almacenan en ellas, deben seguir un modelo lógico adecuado para cumplir su función de manera eficiente. Es decir, de alguna manera hay que indicar a la base de datos qué cosas quiere almacenar, enviando una **descripción** de los datos de interés. Solo así será posible tener un modelo de datos adecuado al problema que se quiera dar solución. Dependiendo de cómo se haya hecho esta descripción de los datos, así cambiarán los procedimientos de acceso, almacenaje y recuperación de los contenidos. Por tanto, dependiendo del modelo de datos utilizado podemos distinguir:

- **BD Jerárquica**: Utilizan árboles para almacenar la información (nodo raíz, nodos padre/hijos, nodos hoja), de manera que la extracción de los datos se realiza mediante un recorrido del árbol y eligiendo la rama por la que debe continuar su búsqueda. Este tipo de estructuras son muy eficientes en las búsquedas pero no pueden reflejar eficientemente la redundancia de datos.
- **BD de Red**: Muy parecida a las jerárquicas pero donde un nodo puede tener varios padres simultáneamente (en contraposición de las bases de datos jerárquicas).
- **BD Relacional**: Es una base de datos que permite resolver el problema que tenían las bases de datos jerárquicas (las redundancias). Mientras que las jerárquicas la posición del dato en el árbol era importante, en las relacionales es irrelevante. Un conjunto de datos compacto y con sentido propio se almacena en algo llamado **tupla**. Una tupla no es más que una fila de una tabla en la que se almacena dicha información compacta. El conjunto de tuplas con información semántica igual determina una tabla. Y un conjunto de tablas pueden ser relacionadas con otras tablas (o tuplas), mediante tablas intermedias que reflejan las relaciones entre cada una de ellas. Esto es lo que soluciona la redundancia de información y son las más utilizadas actualmente.
- **BD Transaccional**: Utilizan el concepto de transacción (que veremos más adelante) que permite asegurar una serie de características importantes en los sistemas informáticos de hoy en día. Estas características se incluyeron en la mayoría de las bases de datos relacionales de hoy en día.
- **BD Multidimensional**: A efectos sería una base de datos Relacional en el que en vez de tener tablas (2D), tendríamos estructuras con N dimensiones para almacenar la información (cubos si fuese 3D).
- **BD Orientadas a Objetos**: Difiere de las bases de datos relacionales en que no se almacena la información por tuplas sino por objetos que contienen la información y los métodos que son necesarios para tratarlas. Es llevar la programación orientada a objetos al mundo de las bases de datos generando características como la herencia, la encapsulación o el polimorfismo.

Como puede observarse, dependiendo del modelo de datos que se quiera almacenar, se deberá utilizar una base de datos u otra.

Independientemente del modelo usado, cuando se tiene un gran conjunto de bases de datos, lo común es utilizar un Sistema de Gestión de Bases de Datos o SGBD, que unifiquen las bases de datos entre sí y se mejoren los flujos de distribución de la información en diferentes entornos y/o formatos.

Para obtener más información, vea la página *http://www.maestrosdelweb.com/principiantes/¿que-son-las-bases-de-datos/*.

Básicamente se deja de utilizar los almacenes de información de las aplicaciones (los ficheros) para utilizar un interfaz de comunicación único (el SGBD y una BD concreta). Su utilización permite:

- ✓ Evitar la redundancia de datos que puede producirse por la duplicación de los mismos ficheros de datos utilizados en distintas aplicaciones y/o ubicaciones.
- ✓ Permiten la abstracción de los datos independientemente del sistema hardware/software utilizado y del soporte utilizado para su almacenamiento.
- ✓ Evitar la inconsistencia de los datos cuando diferentes programas intentan actualizar un dato que es compartido.
- ✓ Evitar que el cambio o adición de nueva información no planeada en el análisis inicial del sistema obligue a la adaptación de todos los programas informáticos que utilizan esos ficheros.
- ✓ Evitar que la mala programación en el acceso/modificación/borrado de los datos en un fichero genere problemas de seguridad. Tener una capa de aplicación y una capa de datos distinta facilita la depuración y la trazabilidad de los programas.

La abstracción, independencia, consistencia, seguridad y accesibilidad son los objetivos que se buscan en estos sistemas. Además su funcionamiento suele ser muy simple. Se puede resumir en un comentario muy acertado de D. Ángel García Moreno, catedrático de la Universidad Politécnica de Madrid y profesor de la asignatura de Bases de Datos en la Ingeniería Informática:

"*Mientras que con los métodos clásicos se accedía a la información a través de ficheros y los datos paseaban por los programas; en los SGBD son los programas los que se dan una vuelta por las Bases de Datos para conseguir lo que necesitan*".

Estos beneficios hacen que la práctica totalidad de los sistemas que quieran almacenar información recurran a un sistema de bases de datos acorde a sus necesidades, garantizando:

- ✓ Gestionar la redundancia para evitarla o al menos limitarla.
- ✓ Mantener los datos separados de las aplicaciones que lo usan.
- ✓ Localizar los datos en una ubicación desde la cual, las aplicaciones puedan acceder de manera distribuida, concurrente y asegurando la integridad de los mismos.
- ✓ Permite realizar una correcta política de copias de seguridad, ya que ya no es necesario ir copiando los ficheros en distintas ubicaciones.

En un SGBD no todo son ventajas. Existen una serie de inconvenientes que es necesario resaltar para tomar en el futuro una decisión adecuada. Como inconvenientes se pueden enumerar los siguientes:

- ✓ Un SGBD no es fácilmente administrable. Requiere de personal especializado en administración de sistemas y lenguajes de consultas contra las bases de datos. Una buena administración de los sistemas y de las bases de datos puede salvar la continuidad del negocio.
- ✓ Si los datos son simples y no requiere de acceso concurrente de usuarios o aplicaciones, puede ser sustituido por un simple fichero, como por ejemplo un archivo binario o un fichero XML.
- ✓ No es sencillo formar al personal que trabaja con los datos almacenados en un SGBD si no tiene un perfil orientado hacia la informática. El personal de nóminas de una empresa, por ejemplo, no tiene porqué aprender un lenguaje de programación de consultas. Requerirá que el sistema le proporcione una interfaz de comunicación tipo formulario de consultas que oculte la complejidad del sistema.
- ✓ Poner en funcionamiento un SGBD requiere de un coste inicial en hardware y software que puede no ser necesario para las necesidades normales de un entorno pequeño.

Siempre habrá que realizar un análisis de los beneficios y los inconvenientes que aportan la integración de estos SGBD en un entorno de trabajo concreto.

Un requisito que hasta el momento no se ha nombrado y que por exigencias del mercado se ha añadido en los SGBD modernos es el soporte transaccional. Una **transacción** es un conjunto de órdenes en secuencia que, en su conjunto, determina un trabajo completo. El trabajo es completo si y solo si se cumplen todas y cada una de sus operaciones en el orden dado. Tras la ejecución de una transacción solo existen dos resultados posibles:

- ✓ Finalizada con éxito.
- ✓ No completada (y asegura que no se ha cambiado nada).

Veamos un ejemplo de cómo de importante es una transacción. Supongamos que un usuario quiere sacar dinero de un cajero automático. La transacción es "*sacar dinero*". Para ello se siguen una serie de pasos secuenciales que permiten realizar la transacción con éxito:

1. Validación del usuario (inserción del PIN).
2. Selección del importe a sacar.
3. Anotación en la cuenta bancaria del reintegro.
4. Expedir tique y el dinero solicitado al usuario.

Si está en el paso número 3 y después hubiera un corte en el suministro eléctrico, el usuario tendría anotada en su cuenta bancaria un reintegro que nunca ha sido entregado al usuario (paso 4). Obviamente el usuario se enfadaría y reclamaría.

Una transacción, si es ejecutada de manera exitosa, asegura que los pasos del 1 al 4 se han seguido en ese orden y que todo ha salido bien. Si la transacción no ha sido exitosa, asegura que ni el dinero ha salido por el cajero, ni que tampoco se ha anotado el reintegro en la cuenta bancario del usuario (incluso aunque haya habido un corte en el flujo eléctrico y estuviese la transacción en el paso 3). Es decir, al no ser exitosa la transacción es como si nunca hubiese pasado.

Esto que, explicado con un ejemplo es tan simple, su nombre técnico en un SGBD se denomina características **ACID**, acrónimo de:

- **Atomicidad** (*Atomicity*): Es la propiedad que asegura que una transacción no se ha quedado a medias. O se ha ejecutado completamente o ninguna de las acciones intermedias ha sido llevada a cabo en la base de datos.
- **Consistencia** (*Consistency*): Una vez se ha determinado que la transacción ha sido exitosa, debe quedar reflejados sus resultados en la base de datos dejando totalmente consistente e integra la base de datos.
- **Aislamiento** (*Isolation*): Una transacción no puede afectar a otra en el transcurso de su ejecución. Esto implica que dos transacciones que trabajen sobre el mismo conjunto de información no puede generar información inconsistente o error alguno en el sistema.
- **Durabilidad** (*Durability*): Una vez realizada la transacción, sus resultados permanecerán inalterables, independientemente de si surgen problemas en el sistema. Esto no quiere decir que una transacción posterior no pueda modificar los datos anteriormente modificados (sí podría pero no de manera concurrente con otra transacción).

Hasta el momento no se ha hablado de cómo está definido un SGBD, por lo que se podría pensar que es un único bloque funcional, hecho *ad hoc*, para resolver un problema de almacenamiento de datos concreto. En realidad un SGBD puede verse como un conjunto de actores y componentes que se intercomunican entre sí para hacer más fácil el acceso a los datos. Estos componentes son:

- **El Hardware**: Dispositivos físicos donde residen todos los demás componentes del SGBD. Normalmente son dispositivos con capacidades de redundancia y con alta disponibilidad para soportar cualquier contingencia que surja durante su funcionamiento.
- **El Software**: Aplicación que permite abstraerse de las características físicas del hardware y que permite hacer al SGBD ser independiente de la plataforma hardware sobre la que se ejecuta.
- **Los Datos**: Información almacenada dentro del recinto físico (hardware) y administrada por el software.
- **Los Usuarios**: Actores a los que se les permite interactuar con el SGBD, haciéndoles transparente el acceso a los datos, independientemente del Hardware o Software utilizado.

En relación a los usuarios, dependerá del software SGBD que se vaya a utilizar pero genéricamente podemos identificar los siguientes roles:

- Administrador del Sistema informático.
- Administrador de la Base de datos y Consultas.
- Usuario Final.

Los anteriores componentes se unen entre sí haciendo que el sistema sea modular y fácil de usar/administrar.

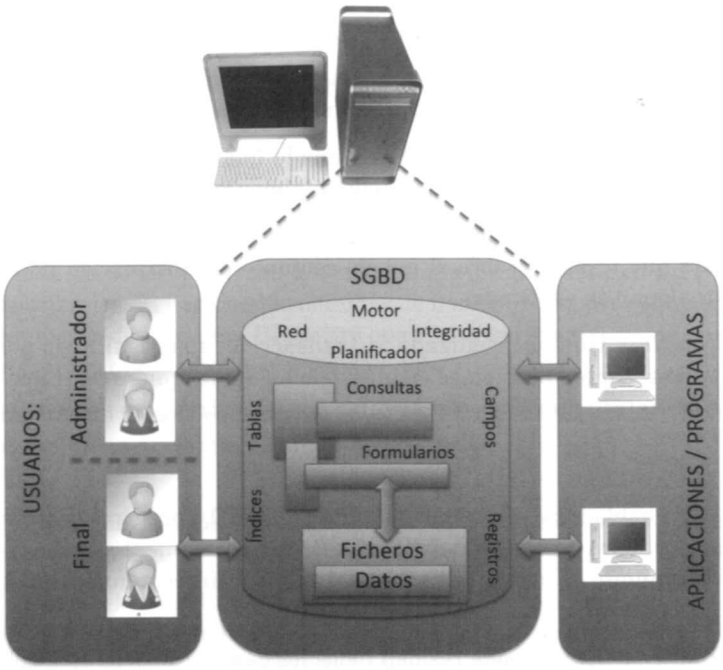

Figura 6.2. *Diagrama de un SGBD*

Únicamente queda por hablar del sistema hardware que se debe utilizar para llevar a cabo la instalación de un SGBD. Para pequeñas y medianas empresas cualquier servidor relativamente moderno puede servir. Todo dependerá del grado de concurrencia de accesos y la cantidad de datos a almacenar los que determinarán las características físicas del hardware. Pero lo que debe ser fundamental es el dimensionamiento del almacén de datos para que no se quede pequeño. Desde la máquina analítica de Charles Babbage hasta los netbook de hoy en día, el almacenamiento de la información ha pasado por muchas etapas:

- El cableado físico.
- La tarjeta perforada.
- La cinta mágnetica.
- El disco duro.
- Los disquetes.
- El CD-ROM / DVD.
- Las tarjetas de memoria.
- Los *pendrives*.
- Los discos duros en estado sólido.

Es claro que los discos duros actuales están alcanzando tamaños impensables, lo que el almacenamiento de datos puede no ser un problema. Simplemente sustituyendo un disco por otro de mayores dimensiones el problema se resuelve y además es seguro que será más rápido que el antiguo (lo que aceleran las transferencias de información).

¿SABÍAS QUE...?

Aunque la información actualmente se almacena de manera digital y se puede acceder a ella de manera muy simple, los sistemas de bases de datos tradicionales no permiten hacer un análisis profundo de las entidades complejas. Es por ello que surge el concepto de las bases del conocimiento (Knowledge Base) que permiten tener consciencia de lo que la base de datos sabe de sí misma a través de relaciones semánticas, ¿será el principio de la inteligencia artificial en las BD?

ACTIVIDADES 6.2

- Estimar, por el método de Fermi, qué tamaño tendrían los datos de la base de datos de un videoclub. Hay que tener en cuenta el espacio requerido para los clientes y para el almacenamiento de los datos de las películas disponibles.
- Buscar en Internet qué es SQL. ¿Para qué sirve?
- Buscar en Internet qué es un **índice** en una BD. ¿Para qué sirve?
- Buscar en Internet y comparar en un cuadro de texto los siguientes tipos de Sistemas Gestores de Bases de Datos (al menos tres de ellos entre sí):
  a. Oracle.
  b. Microsoft SQL Server.
  c. MySQL.
  d. PostgreSQL.
  e. Interbase.
  f. SapDB.
  g. SQLite.

Para obtener más información, vea la página *http://es.wikipedia.org/wiki/Anexo:Comparación_de_sistemas_administradores_de_bases_de_datos_relacionales*.

6.2 UTILIZACIÓN DE XML PARA EL ALMACENAMIENTO DE LA INFORMACIÓN

Gran parte de las BD que hay hoy en día están basadas en un modelo de datos (también llamado modelo de entidad-relación). Es un modelo que ha funcionado bien durante mucho tiempo y que todavía seguirá funcionando. Si bien es cierto que las BD orientadas a objetos permiten simplificar el trabajo de integración con los lenguajes de programación orientados a objetos, cambiar un modelo de trabajo o un sistema que funciona con el modelo anterior es arriesgado. Algunos SGBD tienen modelos híbridos que permiten añadir extensiones al modelo relacional y avanzar a un modelo orientado a objetos de manera poco traumática y de manera transparente.

Tras la llegada de la era Internet, la compartición de la información empezó a resultar crucial para no quedarse desfasado y mejorar las relaciones comerciales. Por ejemplo, una empresa *X* necesita enviarle a la empresa *Y* información en relación a sus transacciones comerciales. El envío de esta información beneficia a *X* y también a *Y* (reducción de costes económicos, costes burocráticos, tiempo, etc.). Es lo que se denomina **B2B** (*Business To Business*). El problema es que *X* utiliza unos modelos de datos distintos a los de *Y*. Tampoco es fácil ponerse deacuerdo con el SGBD. Debe haber algo que permita que ambos sistemas puedan comunicarse entre sí, siendo distintos. La solución es XML.

Para obtener más información, vea la página *http://mymadcat.com/~madcat/talk_tracker.pdf*.

XML permite definir de manera rápida e intuitiva una representación de la información que ambas empresas desean compartir. La empresa *X* usará su SGBD para exportar sus datos a XML y se los remitirá a *Y*. Ambas empresas conocen esa representación de la información por lo que la información fluirá sin problema (independientemente de los campos que tengan sus respectivas BDs o de los SGBD que estén utilizando).

Los SGBD actuales (sobre todo los que usan modelos relacionales), proporcionan en algunos casos extensiones que permitan trabajar con los modelos y representaciones definidas en documentos XML. Aún así, si el objetivo es utilizar XML desde el principio, se debería analizar las siguientes BD XML nativas:

Para obtener más información, vea la página *http://www.info-ab.uclm.es/asignaturas/300219/pdf/BBDDXML.pdf*.

- eXcelon XIS Lite[12].
- TEXTML[13].
- dbXML[14].
- eXist[15].

12 *http://xml.coverpages.org/ExcelonXIS-Lite.html*
13 *http://www.ixiasoft.com*
14 *http://sourceforge.net/projects/dbxml-core/*
15 *http://exist.sourceforge.net/*

Las dos primeras son comerciales y las dos últimas son del mundo OpenSource. Cuando se habla de BD XML nativas, se ha de dejar claro que existen dos maneras almacenar información dentro de ellas:

- **Usando un modelo centrado en el almacenamiento de los datos**: exactamente igual que las BD Relacionales (se guardan tuplas).
- **Usando un modelo centrado en el documento**: no hay campos, ni datos, tal y como se conoce en las BD Relacionales. Se guardan documentos XML.

La primera permite seguir utilizando los modelos relacionales dentro de BD XML. La segunda permite almacenar documentación de diferentes modelos dentro de la BD. Dependiendo de los objetivos de almacenamiento que se planteen quizás se ajuste más un modelo que otro. Ese análisis debe ser meditado concienzudamente.

Dado que el modelo relacional es el más utilizado hoy en día, se indicarán una serie de pasos para usando XML se pueda utilizar el mismo modelo de datos.

Para obtener más información, vea la página *http://www.di.uniovi.es/~labra/cursos/ver06/pres/XMLBD.pdf*.

6.2.1 BASES DE DATOS RELACIONALES

Cuando se utiliza un modelo centrado en el almacenamiento de los datos, la referencia son las BD relacionales. Todo se basa en un conjunto de tablas bidimensionales que permiten almacenar los datos del mundo real.

Para facilitar la comprensión, imaginemos que se quiere representar un conjunto de libros de una biblioteca. Para conseguir almacenar toda la información relativa a cada libro necesitaremos una tabla que almacene atributos como "Título", "Autor", "Editorial", "Edición", "ISBN" y "NumPaginas". Esta tabla se llamaría "Libros". Si se quisiera añadir un libro, se generaría una nueva entrada en la tabla (fila o tupla), en la que se completarán los atributos de ese libro. Si se quisiera añadir un segundo libro a almacenar, se añadiría otra tupla a la tabla. Y así sucesivamente.

TABLA LIBROS

Cód_Libro	Titulo	Autor	Editorial	Edición	ISBN	NumPáginas
1	Don Quijote de la Mancha	Miguel de Cervantes Saavedra	Juan de la Cuesta	3	9788466745840	176
2	La Celestina	Fernando de Rojas	Maxtor	1	9788471664938	320
3	Leyendas	Gustavo Adolfo Bécquer	Cátedra	21	9788437620244	416

Figura 6.3. *Tabla "Libros"*

Antes de continuar añadiendo libros, podríamos darnos cuenta que existen una serie de atributos que podrían ser compartidos entre muchos libros. Sería el caso de "Autor" (un autor puede escribir muchos libros), o "Editorial" (una editorial podría publicar muchos libros de muchos autores distintos), etc. Si por cada libro tuviéramos que rellenar esa misma información, la tabla "Libros" tendría muchísima información redundante. Anteriormente se comentó que el modelo de datos de las BD relacionales hacían una gestión de la redundancia muy eficaz. ¿Cómo resuelve este caso?

Sin duda, si hay información redundante, toda esa información debe salir de la tabla "Libros" y ubicarse en una nueva tabla. Para el caso del atributo "Autor", se podría crear una nueva tabla llamada "Autores" en la que almacenaríamos la información relativa a ellos más un código que le represente de manera única ("CódigoAutor", "Nombre", "Apellidos", "FechaNacimiento", etc.). Con ese código de autor, cada vez que se añada un nuevo libro de un autor ya añadido, únicamente haremos referencia en el campo "Autor" del nuevo libro al código que referencia a dicho autor (en la tabla "Autores"). Esta manera de indireccionar información a otras tablas se denomina **relación**. De la misma manera se haría con el campo "Editorial".

TABLA LIBROS

Cód_Libro	Titulo	Cód_Autor	Editorial	Edición	ISBN	NumPáginas
1	Don Quijote de la Mancha	1	Juan de la Cuesta	3	9788466745840	176
2	La Celestina	2	Maxtor	1	9788471664938	320
3	Leyendas	3	Cátedra	21	9788437620244	416

Figura 6.4. *Tabla "Libros" con relación con "Autores"*

TABLA AUTORES

Cód_Autor	Nombre	Apellidos	Fecha Nacimiento
1	Miguel	de Cervantes Saavedra	29/09/1547
2	Fernando	de Rojas	01/01/1470
3	Gustavo	Adolfo Bécquer	17/02/1836

Figura 6.5. *Tabla "Autores"*

6.2.2 TRANSFORMACIÓN A XML

En principio la adaptación del modelo de datos de una BD relacional a XML es relativamente sencillo de realizar. Una vez obtenido el modelo relacional, como el explicado en el punto anterior, se podría realizar una transformación de tablas a un documento XML simplemente creando una DTD y creando un documento XML bien formado. A continuación se detallarán los pasos a realizar para una tabla de libros y de autores como la anterior:

1 Cada tabla del modelo relacional será un elemento dentro del DTD:

```
<!DOCTYPE Libros[
        <!ELEMENT Libros (libro)*>
]>
```

En este caso se tiene una tabla llamada "Libros" que almacena tuplas de tipo "libro" (cero o más libros).

2 Cada tupla de la tabla se llama "libro". Es necesario indicar qué campos componen a la tupla:

```
<!ELEMENT libro (Cod_Libro, Titulo, Autores, Editorial, Edicion, ISBN, NumPaginas )>
```

3 Cada columna de la tabla deberá establecerse como un tipo de dato almacenable (*char*, *integer*, etc.):

```
<!ELEMENT Cod_Libro(#PCDATA)>
<!ELEMENT Titulo (#PCDATA)>
<!ELEMENT Editorial (#PCDATA)>
<!ELEMENT Edicion (#PCDATA)>
<!ELEMENT ISBN (#PCDATA)>
<!ELEMENT NumPaginas (#PCDATA)>
```

4 Si existe una columna compleja (como puede ser la de "Autores"), se debe crear un nuevo elemento similar al del paso 2. En este caso un libro puede tener uno o más autores (por eso el '+'). También se definen los tipos de datos a almacenar:

```
<!ELEMENT Autores (autor)+>
  <!ELEMENT autor (Cod_Autor, Nombre, Apellidos, FechaNacimiento)>
    <!ELEMENT Cod_Autor (#PCDATA)>
    <!ELEMENT Nombre (#PCDATA)>
    <!ELEMENT Apellidos (#PCDATA)>
    <!ELEMENT FechaNacimiento (#PCDATA)>
```

5 Si dos tablas están relacionadas entre sí a través de una tercera tabla de relación, se establece un nuevo elemento como en el paso 2 y se continua con los pasos siguientes.

La DTD que resulta de todo esto será similar a ésta:

```
<!DOCTYPE Libros[
  <!ELEMENT Libros (libro)*>
    <!ELEMENT libro (Cod_Libro, Titulo, Autores, Editorial,
                Edicion, ISBN, NumPaginas )>
      <!ELEMENT Cod_Libro(#PCDATA)>
      <!ELEMENT Titulo (#PCDATA)>
      <!ELEMENT Editorial (#PCDATA)>
      <!ELEMENT Edicion (#PCDATA)>
      <!ELEMENT ISBN (#PCDATA)>
      <!ELEMENT NumPaginas (#PCDATA)>

      <!ELEMENT Autores (autor)+>
        <!ELEMENT autor (Cod_Autor, Nombre, Apellidos,
                FechaNacimiento)>
          <!ELEMENT Cod_Autor (#PCDATA)>
          <!ELEMENT Nombre (#PCDATA)>
          <!ELEMENT Apellidos (#PCDATA)>
          <!ELEMENT FechaNacimiento (#PCDATA)>
]>
```

Y un ejemplo de documento XML ajustado a la anterior DTD puede ser el siguiente:

```
<?xml version="1.0">
<Libros>
  <libro>
    <Cod_Libro>1</Cod_Libro>
    <Titulo>Don Quijote de la Mancha</Titulo>
    <Editorial>Juan de la Cuesta</Editorial>
```

```
    <Edicion>3</Edicion>
    <ISBN>9788466745840</ISBN>
    <NumPaginas>176</NumPaginas>
    <Autores>
      <autor>
        <Cod_Autor>1<Cod_Autor>
        <Nombre>Miguel</Nombre>
        <Apellidos>de Cervantes Saavedra</Apellidos>
        <FechaNacimiento>29/09/1547</FechaNacimiento>
      </autor>
    </Autores>
  </libro>
  <libro>
    <Cod_Libro>2</Cod_Libro>
    <Titulo>La Celestina</Titulo>
    <Editorial>Maxtor</Editorial>
    <Edicion>1</Edicion>
    <ISBN>9788471664938</ISBN>
    <NumPaginas>320</NumPaginas>
    <Autores>
      <autor>
        <Cod_Autor>2<Cod_Autor>
        <Nombre>Fernando</Nombre>
        <Apellidos>de Rojas</Apellidos>
        <FechaNacimiento>01/01/1470</FechaNacimiento>
      </autor>
    </Autores>
  </libro>
</Libros>
```

6.3 LENGUAJES DE CONSULTA Y MANIPULACIÓN

Tras instalar un SGBD, los usuarios se comunicarán con él mediante algún tipo de lenguaje que permita manipular los datos almacenados. El lenguaje de consulta estructurado más popular es SQL (*Structure Query Language*). Este lenguaje permite, de forma declarativa, acceder a las BD relacionales y operar con ellas. Operaciones típicas son:

- ✓ Creación y borrado de tablas.
- ✓ Inserción, modificación y borrado de tuplas.
- ✓ Ejecución de búsquedas mediante consultas.

SQL es el lenguaje que en la actualidad se considera estándar de facto pues la inmensa mayoría de los SGBD lo implementan. Existen numeroras revisiones del lenguaje (SQL-86, SQL-92, SQL-2003, SQL-2006, SQL-2009) pero la mayoría de los SGBD parten del estándar establecido en 1992. Cabe destacar lo siguiente (en lo relativo a XML se refiere):

- ✓ El estándar 2003 establece ciertas características que permiten un soporte inicial a documentos XML.
- ✓ El estándar 2006 establece una mayor integración con los documentos XML, permitiendo importar y exportar datos en XML. Se establece XQuery como lenguaje de consulta para colecciones de datos en formato XML (y también SQL), aprobado por el W3C (World Wide Web Consortium).

Si, en definitiva, se va a trabajar con documentos XML para almacenar en su interior información mediante un modelo relacional, se necesitará algún lenguaje que permita extraer y manipular información de igual manera que SQL con las BD relacionales. Este lenguaje se llama **XQuery**.

XQuery es, por tanto, un lenguaje de consulta similar a SQL que permite recorrer los documentos XML de manera que se pueda extraer y manipular la información contenida en el mismo. Es un lenguaje muy sencillo que no requiere de conocimientos de programación avanzados (no es necesario saber C/C++, Java, Python, Visual Basic Script, etc). Además XQuery es compatible con muchas de las tecnologías que estandariza W3C (como XML, Namespaces, los esquemas, XSLT y XPath). Es por ello, por su facilidad para obtener resultados sin casi programar y por la compatibilidad con muchos de los estándares actuales, que se ha elegido para realizar las consultas dentro de este libro.

Cuando se va a analizar un documento XML, se crea un árbol de nodos del mismo. Ese árbol tiene un elemento raíz y una serie de hijos. Los hijos del nodo raíz pueden tener más hijos. Si repetimos ese proceso llegará un momento en el que el último nodo no tiene ningún hijo, lo que se denomina nodo hoja.

Establecido ese árbol de nodos, se recorre esa representación del documento XML buscando la infomación que se quiere (ya sea algún nodo concreto, algún atributo de un nodo concreto, etc.). ¿Qué tipos de nodos se puede encontrar en ese recorrido? Básicamente los siguientes:

- **Nodo raíz o "/"**: Es el primer nodo del documento XML. En el ejemplo de la biblioteca de libros explicado anteriormente, sería el elemento "Libros".
- **Nodo elemento**: Cualquier elemento de un documento XML es un nodo elemento en el árbol. El nodo raíz es un caso especial de Nodo elemento (no tiene padre). Cada nodo elemento posee un padre y puede o no poseer hijos. En el caso que no tenga hijos, sería un nodo hoja. En el ejemplo de la biblioteca de libros explicado anteriormente
- **Nodo texto**: Cualquier elemento del documento que no esté marcado con una etiqueta de la DTD del documento XML.
- **Nodo atributo**: Un nodo elemento puede tener etiquetas que complementen la información de ese elemento. Eso sería un nodo atributo.

La extracción de la información durante el recorrido del árbol será tan simple como la detección de los nodos a buscar y el procesamiento de la información que se quiere extraer de ese nodo concreto. Esto que parece tan complicado, se realiza fácilmente con una tecnología denominada **XPath** (XML Path).

XPath es la herramienta que utiliza XQuery para procesar el árbol de nodos de un documento XML. Funciona en base a una serie de expresiones que permiten identificar qué parte del documento XML se quiere acceder o recorrer. La gran ventaja es que al ser una herramienta bastante genérica, XPath no solo es el motor de acceso en documentos XML para XQuery sino que es la base para otras tecnologías como XPointer, XLink o XSLT (como hemos visto anteriormente). Por tanto XQuery usa a XPath para hacer su trabajo.

Pero lo realmente interesante es cómo hacer consultas con XQuery.

¿SABÍAS QUE...?

Los orígenes de SQL están muy relacionados con el desarrollo de las bases de datos relacionales iniciadas en 1970 por Raymond Boyce. Inicialmente se llamaba **SEQUEL** (**S**tructured **E**nglish **Que**ry **L**anguage) y fue diseñado para manipular y extraer datos en el sistema de base de datos diseñado por IBM.

ACTIVIDADES 6.3

- Buscar por Internet que estándar SQL cumplen los siguientes SGBD:
  a. Oracle.
  b. Microsoft SQL Server.
  c. MySQL.
  d. PostgreSQL.
  e. Interbase.
  f. SapDB.
  g. SQLite.
- Buscar por Internet qué es PL/SQL. ¿Para qué sirve?

Para obtener más información, vea la página *http://en.wikipedia.org/wiki/PL/SQL*.

6.3.1 HERRAMIENTA QIZX STUDIO

Qizx Studio[16] es un motor de BD XML que permite el almacenamiento de los documentos XML y la realización de búsquedas y transformaciones a gran velocidad. Qizx es un motor bastante versátil que en la que destacan las siguientes especificaciones técnicas:

Para obtener más información sobre Qizx Studio, vea las páginas *http://kiwi.emse.fr/DN/qizx-manual.pdf* y *http://www.xmlmind.com/qizx/_distrib/docs/manual.pdf*.

16 *http://www.xmlmind.com/qizx/product.html*

Para obtener más información sobre las especificaciones técnicas de Qizx Studio, vea la página *http://www.xmlmind.com/qizx/features.html*.

- ✓ Está orientado a la consulta.
- ✓ No necesita de DTD o esquemas predefinidos, siempre y cuando el documento XML esté bien formado.
- ✓ Realiza un indexado automático lo que permite acelerar las consultas. También puede personalizarse el indexado si se quiere optimizar aún el rendimiento.
- ✓ Permite tener una colección jerárquica de documentos XML (también llamado "XML Library").
- ✓ El almacenamiento de la información se realiza directamente en XML nativo.
- ✓ Soporta XQuery/XPath 2, siendo totalmente compatible con el estándar W3C actual.
- ✓ Permite realizar una representación compacta de la información, comprimiendo tanto los documentos como los índices. De esta manera el almacenamiento puede ser más eficiente y exportarse a otros medios de almacenamiento.
- ✓ Puede funcionar como un anexo a una aplicación propia (embebida) o configurarse como un servidor independiente sobre la que se envían las peticiones de consulta (vía HTTP).
- ✓ Es multiplataforma al haber sido desarrollado por completo en Java. Oficialmente soporta las plataformas:
  - Windows XP, Vista, 7.
  - Linux 2.4+.
  - Mac OS X 10.5+.
- ✓ Necesita de una máquina virtual de Java tipo JRE5+.

Se ha elegido este motor porque tiene la versión Open Source que permitirá practicar con el lenguaje XQuery de manera muy sencilla y sin ningún coste asociado. Esta edición gratuita de Qizx (*Free Engine Edition*) permite tener un límite de tamaño en las BD de 1 GByte, más que suficiente para practicar con XQuery. En cualquier caso, siempre se podría comprar la versión Professional de Qizx si el diseño así lo requiere.

Aunque de ahora en adelante se hablará de Qizx, existen otros desarrollos Open Source, como Apache Lucene[17], que podrían utilizarse para este cometido.

Actualmente está disponible la versión 4.1 de Qizx Free Engine (liberada el 6 de diciembre de 2010). Es posible descargarla en la página web de XMLmind[18]. En caso de utilizar una versión más moderna, los pasos para instalar y ejecutar la BD XML nativa son muy similares a los siguientes:

17 *http://lucene.apache.org*
18 *http://www.xmlmind.com/qizx/download.shtml*

1 Descomprimir el fichero .zip descargado (en caso de que tengamos un entorno Windows).

2 Si se ha descomprimido en "C:\Archivos de Programa\qizx-fe-4.1p1", habrá una carpeta llamada "bin" y dentro un ejecutable llamado "qizxstudio.bat". Hacemos doble clic sobre ese fichero .bat.

3 Aparecerá una ventana parecida a la de la Figura 6.6.

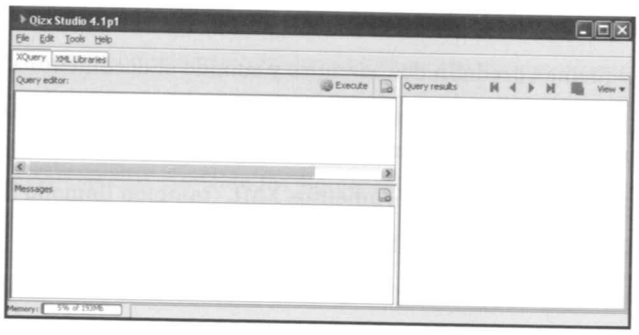

Figura 6.6. *Motor Qizx Free Engine Edition*

Podemos ver lo siguiente:

- Pestaña "XQuery": servirá para insertar las consultas en formato XQuery.
- Pestaña "XML Libraries": indica las librerías de documentos XML cargados en el sistema.
- En la barra de menú, entrada "File": permite crear, abrir y salvar en fichero las consultas XQuery.
- En la barra de menú, entrada "Edit": permite las acciones típicas de deshacer, rehacer, cortar, copiar y pegar.
- En la barra de menú, entrada "Tools": permite abrir un grupo de librerías XML locales, conectarse y desconectarse de una BD XML remota y mostrar logs.
- En la barra de menú, entrada "Help": permite acceder a la ayuda de usuario y a la versión concreta del producto.

Figura 6.7. *Librerías XML cargadas en Qizx*

ACTIVIDADES 6.4

- Buscar por Internet qué estándar XQuery soporta la BD "Oracle XML DB".
- Buscar por Internet que estándar XQuery soporta la BD "BaseX".

Para obtener más información, vea las páginas *http://www.saxonica.com/documentation/about/whatis.xml*, *http://saxon.sourceforge.net/* y *http://download.oracle.com/docs/cd/B28359_01/appdev.111/b28369/xdb_xquery.htm*.

6.3.2 ADMINISTRACIÓN DE LIBRERÍAS XML

Partiendo del documento XML ejemplo que se vió de la biblioteca de libros en la sección "Transformación a XML", vamos a intentar cargar ese fichero como Base de Datos de trabajo:

1 Seleccionar la pestaña "XML Libraries".

2 Botón derecho "Create Library Group". Ver Figura 6.8.

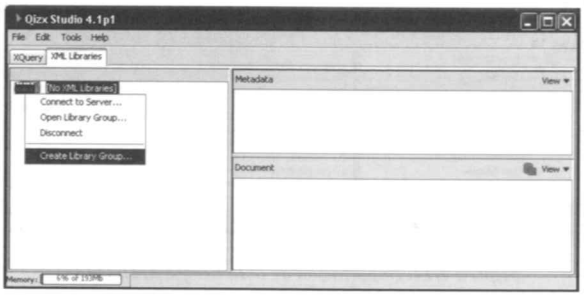

Figura 6.8. *Creación de un grupo de librerías XML*

3 Elegimos un directorio nuevo llamado "GrupoLibreriasXML", donde guardaremos todos nuestros documentos XML y se almacenarán los *logs* de las BD.

4 Botón derecho "Create Library". Ver Figura 6.9.

5 Elegimos un nombre para la BD. En este caso daremos el nombre "BD_Libros". Ver Figura 6.10.

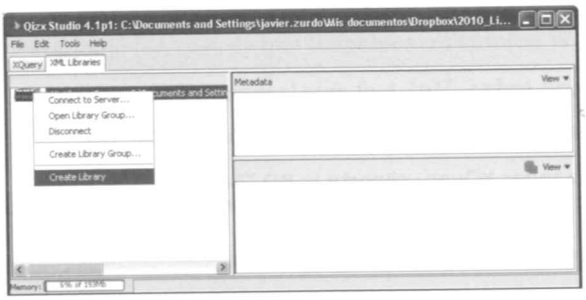

Figura 6.9. *Creación de una librería XML (1/2)*

6 Botón derecho "Import Documents". Ver Figura 6.10.

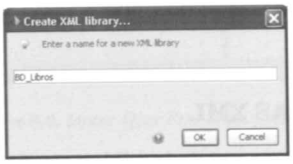

Figura 6.10. *Creación de una librería XML (2/2)*

7 Aparecerá una ventana como en la Figura 6.11. Pulsar el botón "Add File/Folder ...".

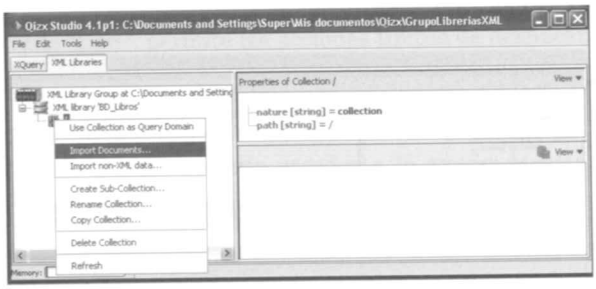

Figura 6.11. *Importación de documentos XML (1/4)*

8 Seleccionar el documento XML que se vió en la sección anterior. Llamarlo "BD_Libros.xml" y pulsar el botón "Start Import". Ver Figura 6.12. Si la importación ha sido exitosa, pulsar el botón "Close".

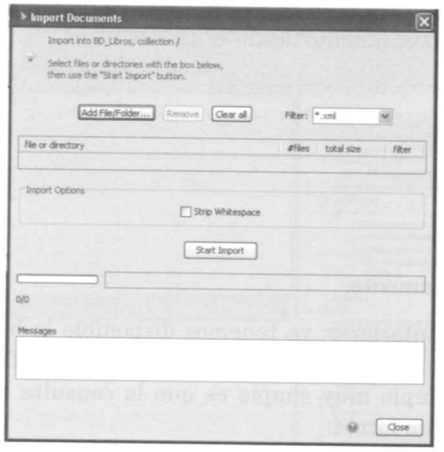

Figura 6.12. *Importación de documentos XML (2/4)*

9 Tras la importación, el documento quedará vinculado a la librería "BD_Libros" y dentro del grupo de librerías "GrupoLibreriasXML". Ver Figuras 6.13 y 6.14.

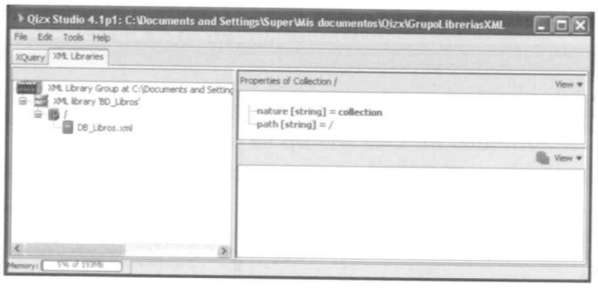

Figura 6.13. *Importación de documentos XML (3/4)*

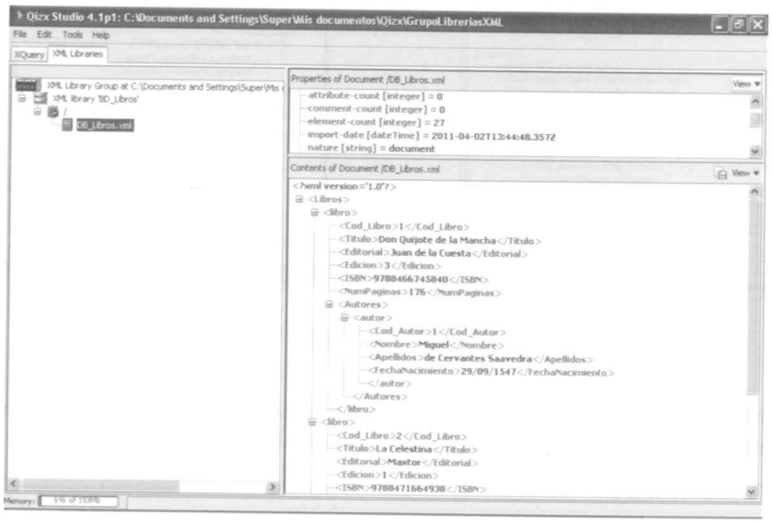

Figura 6.14. *Importación de documentos XML (4/4)*

Ya estaría disponible en la base de datos el documento XML. Si quisiéramos importar otros documentos en la misma librería, se debería repetir el procedimiento desde el paso 6.

6.4 XQUERY

Se ha hablado de XQuery muy ligeramente.

Si se han seguido todos los pasos anteriores, ya tenemos disponible la base de datos para hacer consultas. En primer lugar se va a utilizar una función de XQuery llamada "doc(<nombreDocumento.xml>)", que permite extraer datos del documento indicado. Un ejemplo muy simple es que la consulta nos devuelva todo el documento XML almacenado. Esto se haría de la siguiente forma:

doc("BD_Libros.xml")

Habiendo escrito esta función en la pestaña "XQuery", pulsamos el botón "Execute" y veremos en la parte inferior ("Messages"), si se ha ejecutado correctamente y el tiempo empleado para ello. Pero lo realmente interesante es el resultado de la consulta, que ha sido devuelta en la parte derecha de la aplicación.

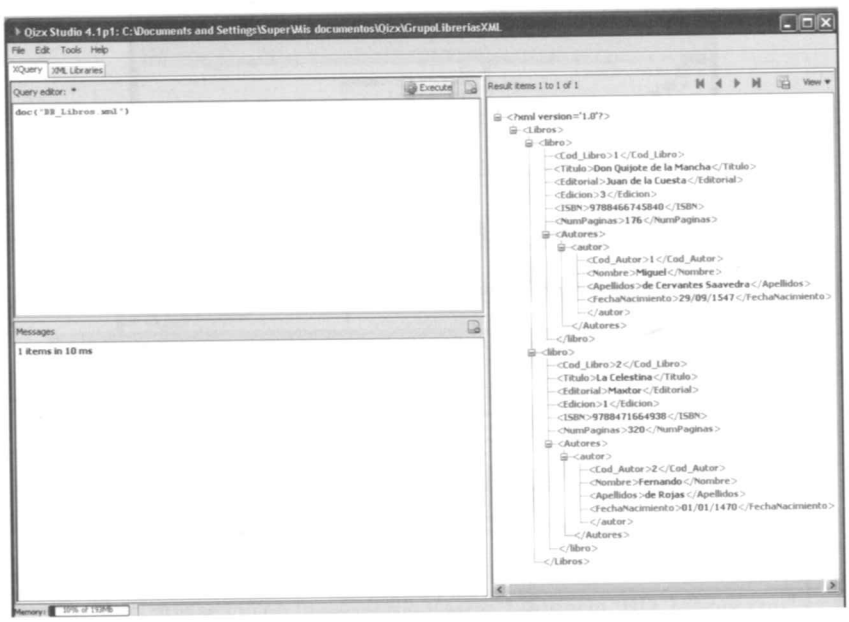

Figura 6.15. *XQuery básica: Mostrar todo el documento XML*

Si en vez de devolver todo el documento, se quiere acceder únicamente a un conjunto de nodos bien identificados, añadiríamos el camino en el árbol para acceder a ellos, por ejemplo los libros:

doc("BD_Libros.xml")//Libros/libro

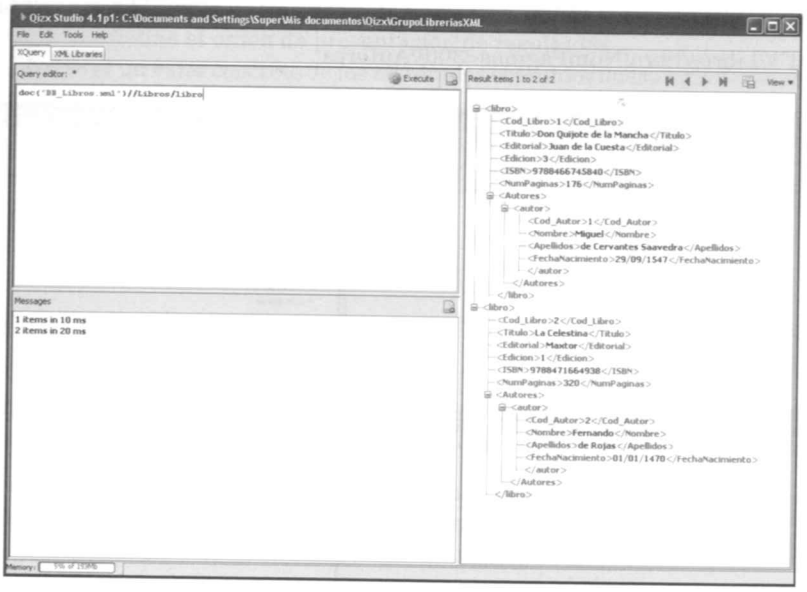

Figura 6.16. *XQuery básica: Mostrar todos los libros*

Como se puede observar, aparecen todos los libros que había almacenados en el documento XML inicial. Al igual que con las expresiones XSLT es posible que se quiera solo un conjunto de nodos dependiendo de un patrón de búsqueda. Si queremos todos los libros que tengan menos de 300 hojas la consulta podríamos completarla de la siguiente manera:

doc("BD_Libros.xml")//Libros/libro[NumPaginas<300]

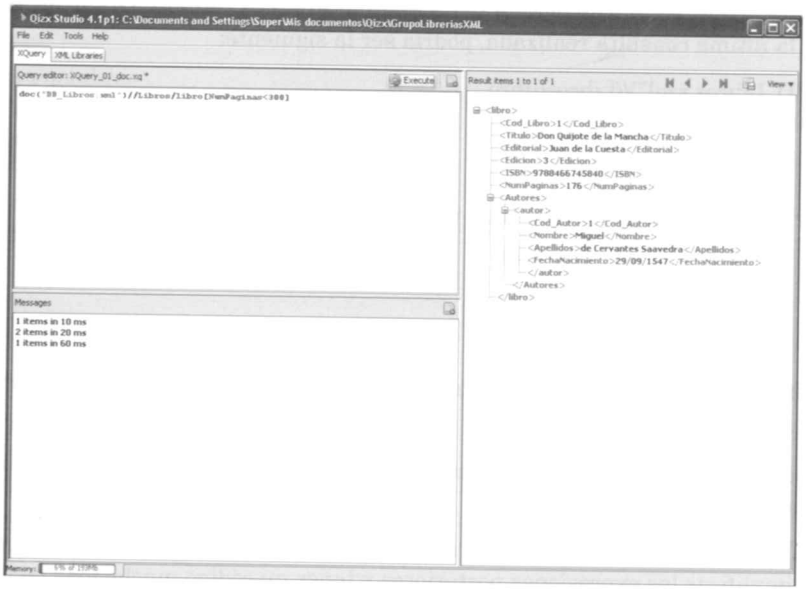

Figura 6.17. *XQuery básica: Mostrar libros con menos de 300 páginas*

¿Y si solo queremos los autores de los libros que tienen menos de 300 páginas? La consulta sería ésta:

doc("BD_Libros.xml")//Libros/libro[NumPaginas<300]/Autores

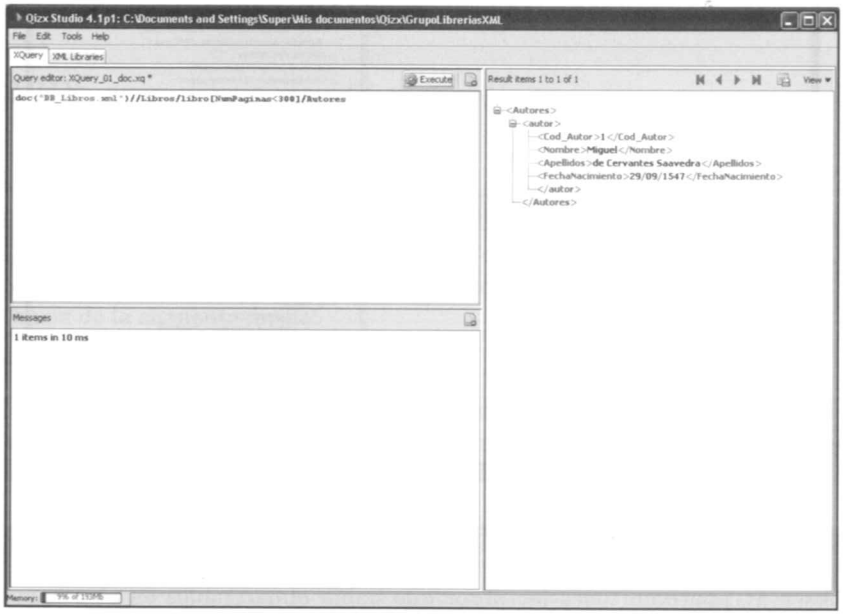

Figura 6.18. *XQuery básica: Mostrar autores con libros de menos de 300 páginas*

Los ejemplos anteriores son la manera sencilla de realizar búsquedas y selecciones de nodos concretos en un documento XML. Pero existe otra manera, mucho más potente, para realizar este trabajo. Es lo que se denomina **expresiones FLWOR**. FLWOR es la contracción del acrónimo **F**or, **L**et, **W**here, **O**rder by, **R**eturn. Una expresión FLWOR equivalente a la última consulta realizada, podría ser la siguiente:

for $libro in doc("BD_Libros.xml")//Libros/libro
where $libro/NumPaginas<300
return $libro/Autores

A continuación se explicará para qué sirven cada una de las clausulas FLWOR:

Para obtener más información, vea la página *http://www.stylusstudio.com/xquery_flwor.html*.

- **for**: Esta sentencia permite seleccionar los nodos que se quieren consultar, guardándose en la variable (el identificador que le precede el símbolo $).
- **let**: Esta clausula es opcional. Esta sentencia establece una nueva variable sobre el mismo u otro documento XML. Permite simplificar las expresiones posteriores y tener un código mucho más legible.

- **where**: Clausula que permite establecer una condición sobre la variable indicada en "for" y "let".
- **order by**: Clausula que define el orden de presentación de resultados.
- **return**: Permite devolver un valor concreto de los resultados obtenidos de las anteriores clausulas (uno por nodo).

Cabe destacar que la utilización de expresiones FLWOR resulta tremendamente similar a las consultas realizadas en SQL en las bases de datos relacionales. Usaremos estas expresiones para extraer la información de nuestras bases de datos.

6.5 CONSULTAS

Antes de continuar, se va a crear un nuevo documento XML y se añadirá en la base de datos de Qizx Studio. La razón fundamental es que necesitamos documento más denso en datos para poder realizar ejemplos complejos.

Imaginemos que tenemos un conocido que se gana la vida con una academia de bailes de salón. Nos pide ayuda para almacenar información que considera fundamental en su negocio. Estos datos son:

- Nombre del baile.
- Precio de la clase (indicando la periodicidad de la cuota y la moneda de pago).
- Numero de plazas disponibles.
- Fecha de comienzo de las clases.
- Fecha de finalización de las clases.
- Nombre del profesor que la imparte.
- Sala en la que se desarrollará la clase.

Tras un período recogiendo los datos y plasmándolos en XML, el documento queda así:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Bailes>
  <baile id="1">
    <nombre>Tango</nombre>
    <precio cuota="mensual" moneda="euro">27</precio>
    <plazas>20</plazas>
    <comienzo>1/1/2011</comienzo>
    <fin>1/12/2011</fin>
    <profesor>Roberto Garcia</profesor>
    <sala>1</sala>
  </baile>

  <baile id="2">
    <nombre>Cha-cha-cha</nombre>
    <precio cuota="trimestral" moneda="euro">80</precio>
    <plazas>18</plazas>
    <comienzo>1/2/2011</comienzo>
    <fin>31/7/2011</fin>
    <profesor>Miriam Gutierrez</profesor>
    <sala>1</sala>
  </baile>
```

```
<baile id="3">
  <nombre>Rock</nombre>
  <precio cuota="mensual" moneda="euro">30</precio>
  <plazas>15</plazas>
  <comienzo>1/1/2011</comienzo>
  <fin>1/12/2011</fin>
  <profesor>Laura Mendiola</profesor>
  <sala>1</sala>
</baile>

<baile id="4">
  <nombre>Merengue</nombre>
  <precio cuota="trimestral" moneda="dolares">75</precio>
  <plazas>12</plazas>
  <comienzo>1/1/2011</comienzo>
  <fin>1/12/2011</fin>
  <profesor>Jesus Lozano</profesor>
  <sala>2</sala>
</baile>

<baile id="5">
  <nombre>Salsa</nombre>
  <precio cuota="mensual" moneda="euro">32</precio>
  <plazas>10</plazas>
  <comienzo>1/1/2011</comienzo>
  <fin>1/12/2011</fin>
  <profesor>Jesus Lozano</profesor>
  <sala>2</sala>
</baile>

<baile id="6">
  <nombre>Pasodoble</nombre>
  <precio cuota="anual" moneda="euro">320</precio>
  <plazas>8</plazas>
  <comienzo>1/1/2011</comienzo>
  <fin>31/12/2011</fin>
  <profesor>Miriam Gutierrez</profesor>
  <sala>2</sala>
</baile>
</Bailes>
```

La inclusión de este documento en la base de datos Qizx es tan sencilla como ir a la pestaña "XML Libraries", seleccionar como "XML Library Group" la "GrupoLibreriasXML". Después seleccionar "BD_Libros" como "XML library". Y en el raíz de BD_Libros, botón derecho "Import Documents ...". El procedimiento es igual que el indicado en el apartado 6.3.2.

Una vez cargado en la base de datos, podemos volver a la pestaña "XQuery" y realizar las consultas. Establezcamos las siguientes consultas para practicar:

EJEMPLO 6.1

Se necesita saber qué bailes se realizan en la sala número 1:

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
where $baile/sala = 1
return $baile/nombre
```

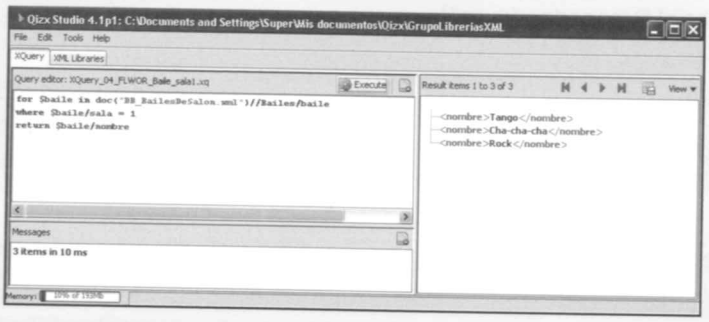

Figura 6.19. FLWOR: Nodos de bailes impartidos en la sala 1

Como puede observarse en la Figura 6.19, los resultados salen con las etiquetas de XML. Si quisiéramos solo tener un listado, obviando las etiquetas "<nombre>" y "</nombre>", simplemente tendríamos que indicar en la clausula "return" que extraiga los datos de ese elemento XML:

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $n:=$baile/nombre
where $baile/sala = 1
return data($n)
```

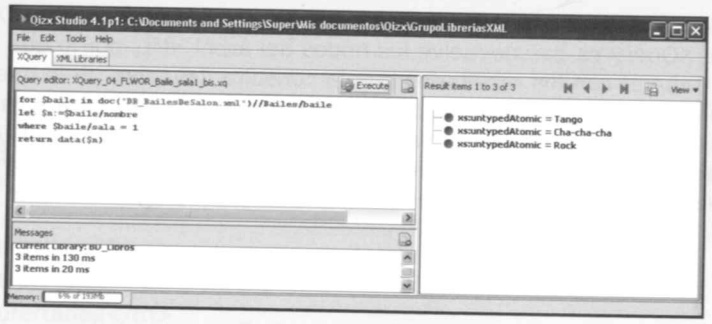

Figura 6.20. FLWOR: Datos de los nodos de bailes impartido en la sala 1

Tal y como se ve en la Figura 6.20, los resultados ya no están con las etiquetas del elemento "nombre". Todo es debido a que existe una función llamada "data(<variable>)", que extrae del nodo la información almacenada.

EJEMPLO 6.2

Se necesita extraer los nodos de aquellos bailes que se impartan en la sala número 2 y cuyo precio sea menor que 35 euros:

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $n:=$baile/nombre
where $baile/sala = 2 and $baile/precio < 35
    and $baile/precio[@moneda="euro"]
return $n
```

El resultado será:

```
<nombre>Salsa</nombre>
```

EJEMPLO 6.3

Se necesita saber el nombre de los profesores que dan clases con cuotas mensuales:

```
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $profesor:=$baile/profesor
where $baile/precio[@cuota="mensual"]
return $profesor
```

El resultado será:

```
<profesor>Roberto Garcia</profesor>
<profesor>Laura Mendiola</profesor>
<profesor>Jesus Lozano</profesor>
```

Estos tres ejemplos han permitido descubrir las grandes similitudes que tiene XQuery con SQL. Una de las ventajas de XQuery es que devuelve los nodos del árbol XML, por lo que cualquier programa o aplicación puede trabajar con los resultados que la consulta devuelva. Si lo que se quiere es extraer solo la información y no los nodos, hemos descubierto que la función "data(<variable>)" nos permite conseguirlo. Pero, ¿y si lo que se quiere es devolver un fichero HTML con los datos de la consulta? Quizás el lector recuerde que con XSL se podía tratar este caso. A continuación veremos como generar HTML con una consulta a una base de datos XML nativa, mediente expresiones FLWOR.

6.5.1 DE FLWOR A HTML

Queremos crear una consulta XQuery que tras ejecutarla nos devuelva los resultados en formato HTML. Podemos unir dentro de XQuery etiquetas HTML y expresiones o clausulas FLWOR. La única limitación es que cuando se fusionan en una consulta, indiquemos al motor de consultas qué parte es la que tiene que procesar como consulta. Para ello indicaremos entre llaves ("{", "}"), que parte es FLWOR. La mejor manera de verlo es a través de un par de ejemplo (se marcará en negrita la parte de la consulta FLWOR):

EJEMPLO 6.4

Queremos una consulta XQuery cuyo resultado sea una tabla HTML que nos muestre el nombre del baile, el profesor que lo imparte y el número de plazas ofertadas:

```
<html>
<body>
  <h1> Bailes ofertados </h1>
    <table border="1">
      <tr>
        <th>Nombre baile</th>
        <th>Nombre profesor</th>
        <th>Plazas ofertadas</th>
      </tr>
{
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $nombre:=$baile/nombre
let $profesor:=$baile/profesor
let $plazas:=$baile/plazas
return
    <tr>
      <td>{data($nombre)}</td>
      <td>{data($profesor)}</td>
      <td>{data($plazas)}</td>
    </tr>
}
    </table>
</body>
</html>
```

El resultado es:

```
<html>
<body>
 <h1> Bailes ofertados </h1>
 <table border="1">
   <tr>
     <th>Nombre baile</th>
     <th>Nombre profesor</th>
     <th>Plazas ofertadas</th>
   </tr>
   <tr>
     <td>Tango</td>
     <td>Roberto Garcia</td>
     <td>20</td>
   </tr>
```

```
    <tr>
      <td>Cha-cha-cha</td>
      <td>Miriam Gutierrez</td>
      <td>18</td>
    </tr>
    <tr>
      <td>Rock</td>
      <td>Laura Mendiola</td>
      <td>15</td>
    </tr>
    <tr>
      <td>Merengue</td>
      <td>Jesus Lozano</td>
      <td>12</td>
    </tr>
    <tr>
      <td>Salsa</td>
      <td>Jesus Lozano</td>
      <td>10</td>
    </tr>
    <tr>
      <td>Pasodoble</td>
      <td>Miriam Gutierrez</td>
      <td>8</td>
    </tr>
  </table>
</body>
</html>
```

Bailes ofertados

Nombre baile	Nombre profesor	Plazas ofertadas
Tango	Roberto Garcia	20
Cha-cha-cha	Miriam Gutierrez	18
Rock	Laura Mendiola	15
Merengue	Jesus Lozano	12
Salsa	Jesus Lozano	10
Pasodoble	Miriam Gutierrez	8

Figura 6.21. *FLWOR: Tabla HTML con baile, profesor y plazas*

Como puede observarse en la Figura 6.21, el resultado es totalmente compatible con HTML y visible en cualquier navegador actual.

EJEMPLO 6.5

Queremos realizar la misma consulta anterior pero estableciendo la condición de ser bailes con cuota trimestral. Además se ordenará, de menor a mayor, los bailes según el número de plazas disponibles. La consulta XQuery es exactamente igual que antes pero cambiando únicamente la parte FLWOR. Remarcamos en negrita la diferencia con respecto a lo anterior:

```
{
for $baile in doc("DB_BailesDeSalon.xml")//Bailes/baile
let $nombre:=$baile/nombre
let $profesor:=$baile/profesor
let $plazas:=$baile/plazas
where $baile/precio[@cuota = "trimestral"]
order by $baile/plazas
return
    <tr>
      <td>{data($nombre)}</td>
      <td>{data($profesor)}</td>
      <td>{data($plazas)}</td>
    </tr>
}
```

El resultado es:

```
<html>
<body>
 <h1> Bailes ofertados </h1>
 <table border="1">
   <tr>
     <th>Nombre baile</th>
     <th>Nombre profesor</th>
     <th>Plazas ofertadas</th>
   </tr>
   <tr>
     <td>Merengue</td>
     <td>Jesus Lozano</td>
     <td>12</td>
   </tr>
   <tr>
     <td>Cha-cha-cha</td>
     <td>Miriam Gutierrez</td>
     <td>18</td>
   </tr>
</table>
</body>
</html>
```

Bailes ofertados

Nombre baile	Nombre profesor	Plazas ofertadas
Merengue	Jesus Lozano	12
Cha-cha-cha	Miriam Gutierrez	18

Figura 6.22. FLWOR: Tabla HTML con bailes con cuota trimestral

Puede verse el resultado en el navegador web en la Figura 6.22.

6.6 ACTUALIZACIÓN

Hasta el momento solo se ha podido realizar consultas sobre los contenidos ya almacenados en la base de datos XML. Quizás nuestro amigo quiera insertar, reemplazar, renombrar, cambiar y/o borrar entradas dentro de la base de datos. Para ello indicaremos qué hacer en cada caso.

Para obtener más información, vea las páginas *http://www.xmlplease.com/xquery-update*, *http://exist.sourceforge.net/update_ext.html* y *http://stackoverflow.com/questions/ 5335046/xquery-update-insert-or-replace-depending-if-node-exists-not-possible*.

6.6.1 INSERCIÓN

EJEMPLO 6.6

Se quiera añadir un nuevo baile en la base de datos. Los datos son:

- Nombre: Foxtrot.
- Precio: 22 dólares.
- Pago: mensual.
- Plazas: 12.
- Comienzo: 01/01/2012.
- Fin: 31/07/2012.
- Profesor: Freddy Astaire.
- Sala: 3.

La inserción se realizaría de la siguiente manera:

```
insert node
<baile id="7">
      <nombre>Foxtrot</nombre>
      <precio cuota="mensual" moneda="dolares">22</precio>
      <plazas>12</plazas>
      <comienzo>01/01/2012</comienzo>
      <fin>31/07/2012</fin>
      <profesor>Freddy Astaire</profesor>
      <sala>3</sala>
</baile>
before doc("DB_BailesDeSalon.xml")//Bailes/baile[1]
```

Este código inserta el nodo indicado en la base de datos "DB_BailesDeSalon.xml". El nodo se insertará antes del primer nodo de la base de datos (por la clausula "before"). Si se quiere insertar después, solo cambiaría "before" por "after". Ambas utilizan como referencia al primer nodo ("[1]").

Una sentencia equivalente para insertar al principio de la base de datos, sin tener que referenciar a nodo alguno de la base de datos, sería sustituyendo la última línea por esta otra:

as first into doc("DB_BailesDeSalon.xml")//Bailes

Si por el contrario, lo que se desea es insertar al final de la base de datos, sin referenciar a ningún nodo, se realizaría de la siguiente manera:

as last into doc("DB_BailesDeSalon.xml")//Bailes

6.6.2 REEMPLAZO

EJEMPLO 6.7

En la inserción anterior se cometieron dos errores:

1. El nombre correcto era "Angel Correllada".
2. El número de plazas realmente eran 14.

Si en la inserción anterior se realizó antes del primer nodo de la base de datos, entonces el elemento insertado será ahora el primero. Se cambiarán esos datos de dos maneras distintas:

- Mediante la modificación del valor del nodo.
- Mediente el reemplazo del nodo completo.

La modificación se realizaría de la siguiente manera:

```
replace value of node
doc("DB_BailesDeSalon.xml")//Bailes/baile[1]/profesor
with "Angel Correllada"
,
replace node
doc("DB_BailesDeSalon.xml")//Bailes/baile[1]/plazas
with <plazas>14</plazas>
```

Como puede verse, las dos modificaciones se han realizado a la vez pero separándolas por una coma ",". En el caso de no saber en qué posición de la base de datos se ha realizado la inserción (en principio debe dar igual la posición mientras esté insertados los datos), se podría tener un problema de actualizar erróneamente la tupla incorrecta. Para evitar este problema, lo más común es realizar la modificación mediante la utilización del identificador del baile (en este caso id=7).

```
replace value of node
doc("DB_BailesDeSalon.xml")//Bailes/baile[@id=7]/profesor
with "Angel Correllada"
,
replace node
doc("DB_BailesDeSalon.xml")//Bailes/baile[@id=7]/plazas
with <plazas>14</plazas>
```

6.6.3 BORRADO

EJEMPLO 6.8

Después de la inserción y la modificación del nuevo baile en la base de datos, finalmente parece que no se va a desarrollar ese curso. Se pide que finalmente se borre esa tupla en la base de datos (id=7):

```
delete node doc("DB_BailesDeSalon.xml")//Bailes/baile[@id=7]
```

6.7 EXPORTACIÓN DE LIBRERÍAS XML

Después de realizar todos los cambios en la base de datos, es un buen momento para exportar el contenido a un fichero XML externo. De esta manera se podría tener una copia de seguridad en caso de contingencia. Qizx Studio nos permite realizarlo de manera muy sencilla:

1. Ir a la pestaña "XML_Libraries".
2. Seleccionar "DB_BailesDeSalon.xml", botón derecho y pulsar "Export to File ...".
3. Aparece una nueva ventana en la que se puede seleccionar el fichero de salida, la codificación y el formato (XML, HTML, XHTML, o texto). Para hacer una copia de seguridad se selecciona XML y se prodecerá a volcar la información. Ver Figura 6.23.

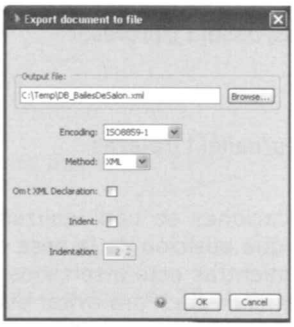

Figura 6.23. *Exportación de librerías XML en Qizx*

Para finalizar, se habló anteriormente de cómo utilizar expresiones FLWOR para que devuelvan código HTML. El procedimiento para exportar los resultados es muy sencillo. Una vez que se está en la pestaña "XQuery" y se ejecuta la consulta, en la parte derecha muestra los resultados. Hay que pusar el botón de exportación de resultados "Save results" y aparecerá una ventana similar a la de la Figura 6.23. De esta manera tendremos en fichero los resultados de la consulta.

6.8 OTRAS FUNCIONES O LIBRERIAS

Se ha visto que XQuery[19] es una herramienta de alto nivel que permite realizar todas las tareas administrativas de un SGBD sin tener que programar ninguna línea en C/C++ o Java. Pero XQuery no es solamente lo que se ha visto aquí. Recordemos que XQuery usa por XPath y éste tiene muchas funcionalidades que no se han tratado en este capítulo. Existen funciones para acceder a nodos padre, funciones para tratamiento de valores numéricos, para tratar strings, resolver URIs, funciones para tratar fechas/horas, para acceder a nodos, para establecer secuencias... En el caso de necesitar más potencia a la hora de generar las consultas, se recomienda consultar las funciones de XPath[20].

Otra manera de acceder a los nodos de los documentos XML es mediante la librería **DOM**[21] (*Document Object Model*). DOM es una interfaz de programación de aplicaciones (API), que permite representar documentos HTML y/o XML de manera que las aplicaciones que la utilicen puedan representar el documento en forma de árbol. Una vez cargado el árbol, se puede recorrer buscando información, modificando los propios nodos o la información que contienen, cambiando la estructura del árbol, etc. Esta API es un estándar de acceso aprobada por la W3C. El inconveniente es que es necesario conocer un lenguaje de programación para hacer uso de esta API. En la actualidad la práctica mayoría de los navegadores la tienen implementada.

Por último tenemos otra API llamada **SAX**[22] (*Simple API for XML*). Es una librería que inicialmente se utilizaba en Java pero que en la actualidad es posible utilizarla en muchos lenguajes de programación. La principal ventaja de usar un parser SAX es que el documento XML se lee una única vez, de manera secuencial y sin que se cargue todo el en memoria (sin generar ningún árbol). Esto redunda en que los analizadores SAX son más eficientes en la memoria utilizada que los DOM. Como desventaja es que una vez pasada la lectura, no se puede volver a atrás (en contraposición de DOM que sí se permite).

En resumen, existen muchas maneras de manipular documentos XML. En este libro se ha elegido XQuery pues permite utilizarse en los grandes sistemas SGBD y también en pequeñas aplicaciones con motores de BD XML nativas como Qizx. No será necesario tener conocimientos de programación en lenguajes como C/C++ o Java para manejar API's como SAX o DOM. En cualquier caso, la decisión final de utilizar una u otra solución dependerá de las necesidades que tenga el proyecto al que vaya a implantarse, haciendo especial hincapié en el almacenamiento y el tratamiento de la información.

6.9 CASO PRÁCTICO

Se propone analizar y diseñar un caso concreto de almacenamiento de información en bases de datos. En este caso se dará libertad al alumno (bajo supervisión del profesor) del sistema a modelar. Podrían ser casos como:

- Una mediateca.
- Almacenamiento de transacciones bancarias.
- Almacenamiento de mensajes de texto recibidos de publicidad en TV.
- Almacenamiento de ofertas de hoteles y viajes.
- Etc.

19 *http://www.w3.org/TR/xquery/*
20 *http://www.w3.org/TR/xpath/*
21 *http://www.w3.org/DOM/*
22 *http://www.saxproject.org/*

Se realizarán todos los pasos necesarios para pasar de un modelo de datos relacional a un modelo en base de datos XML nativa. Se generará un documento XML que almacene dicha información y se realizarán, al menos dos consultas para:

- Inserción.
- Modificación.
- Borrado.
- Consulta de información con presentación de resultados en HTML.

RESUMEN DEL CAPÍTULO

En este capítulo se ha pretendido poner en conocimiento la técnica Fermi para la aproximar las necesidades de un entorno cuando este último no tiene datos suficientes para ello. Esta técnica no está explícitamente explicada pero se propone como actividad por sus grandes resultados en las estimaciones de cualquier disciplina (y no solo en la estimación del almacenamiento requerido en una base de datos).

Se ha puesto de manifiesto qué tipos de bases de datos existen en la actualidad y se han relacionado entre sí mostrando las fortalezas y las debilidades de cada una de ellas. Se ha introducido el concepto de Sistemas de Gestión de Bases de Datos como el elemento central en la administración de las bases de datos. Se incide en la importancia real de las transacciones y en las características que se deben implantar en los SGBD para impedir la pérdida de información (ACID).

Se ha relacionado SQL y XQuery como lenguajes que permiten realizar las búsquedas, inserciones, borrados o modificaciones de las tuplas, tablas o bases de datos en sistemas relacionales y sistemas nativos XML respectivamente. Además se ha proporcionado un método para realizar la conversión de un modelo de datos relacional a un sistema basado en documentos XML para su inclusión en una BD XML nativa.

Se ha proporcionado los elementos básicos para manejar una BD XML nativa, como Qizx Studio sin que el usuario tenga que realizar un desembolso económico elevado para practicar las consultas XQuery. Además se ha enseñado como con las expresiones FLWOR se permite generar como resultados documentos XML o HTML utilizables en otras aplicaciones externas.

Para finalizar se ha comparado someramente como otras APIs permiten realizar una funcionalidad similar a XQuery pero necesitando conocimientos de lenguajes de programación como C/C++ o Java para tener acceso a todos los nodos del documento XML.

EJERCICIOS PROPUESTOS

- **1.** ¿A qué científico debemos la posibilidad de realizar estimaciones muy precisas a problemas con falta de datos suficientes? Pista, buscar en Internet por "cuestiones de Fermi".
- **2.** Buscar por Internet los siguientes conceptos:
  - B2B.
  - B2C.
  - B2E.

  Realizar una tabla en la que se muestren las ventajas de la aplicación de estas estrategias comerciales en una empresa.
- **3.** Añadir en la DTD de la biblioteca de libros, la nacionalidad del autor. Completar el nuevo campo en el documento XML que los almacena.
- **4.** Añadir en el documento XML 10 nuevos libros.
- **5.** Realizar una consulta XQuery con expresiones FLWOR que muestre todos los libros almacenados.
- **6.** Realizar una consulta XQuery con expresiones FLWOR que muestre todos los libros almacenados con más de 500 páginas.
- **7.** Realizar una consulta XQuery con expresiones FLWOR cuyo resultado sea una tabla HTML con todos los libros de la biblioteca.
- **8.** Realizar una consulta XQuery con expresiones FLWOR cuyo resultado sea una tabla HTML con todos los libros de la biblioteca siempre que tengan más de 150 páginas.
- **9.** Realizar una inserción de un nuevo libro con XQuery (con algún dato erróneo).
- **10.** Realizar una modificación del libro anterior corrigiendo el dato erroneo.
- **11.** Realizar el borrado del último libro insertado en la base de datos con XQuery.