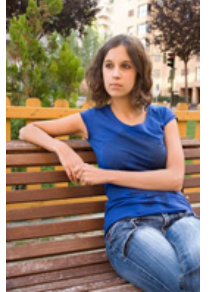


Caso práctico

A la empresa BK Programación no dejan de llegar nuevos proyectos, está claro que sus productos finales convencen a sus clientes y éstos están bastante contentos con ellos.

Ana está orgullosa de haber participado y aportado su granito de arena en el desarrollo y mantenimiento de diferentes aplicaciones, algunas bastante complejas.

Este fin de semana se ha tomado un respiro. Sabe que el lunes comienza con algo nuevo. Debe ayudar a **María** y **Juan** en la tarea de compatibilizar diferentes formatos de datos que intercambian varias academias de Inglés de una misma firma.



Las academias gestionan la información relativa a sus cursos, alumnos, profesores, distribuidores de material, libros de lectura, artículos, apuntes, exámenes, etc. con programas diferentes y basados también en sistemas de bases de datos diferentes.

Hasta ahora, las academias han hecho uso de XML, para intercambiar los datos entre sus sistemas, pero cada vez surge algún problema nuevo, sobre todo con el almacenamiento, consulta y recuperación de ciertos documentos de texto. Por tanto, han decidido acudir a BK Programación para buscar la mejor solución. No les importa empezar desde cero y, si es necesario, utilizar una base de datos nativa XML, tal y como les ha sugerido **Ada** en su primera entrevista.

Ana apura la tarde del domingo pues esa misma noche va a comenzar su nueva aventura, ¡repasar XML y las tecnologías relacionadas! **Ana**, es muy profesional y no deja para mañana lo que puede hacer hoy.

1.- Introduccion.

Atendiendo al nivel de estructuración, podemos decir que existen tres tipos de datos:

- ✓ **Datos estructurados.** Son los que tienen un formato estricto. Toda la información recogida se ajusta al mismo formato, como por ejemplo: los datos tabulados en filas y columnas de una tabla.
- ✓ **Datos desestructurados.** No tienen ninguna estructura, como un documento de texto o un archivo de vídeo.
- ✓ **Datos semi-estructurados.** Tienen cierta estructura, pero no toda la información recogida tiene la misma forma, y además puede ir variando de manera dinámica.

The logo for XML (eXtensible Markup Language) is displayed in a stylized, bold, orange font with a black outline, set against a light blue rectangular background.

Las bases de datos tradicionales, como las Bases de Datos Relacionales (BDR) son apropiadas para almacenar datos estructurados, pero la cuestión es que, en la actualidad, son muchas las situaciones en las que interesa almacenar grandes volúmenes de datos no estructurados, e incluso integrarlos con datos estructurados. Pero ¿cómo hacerlo? Aquí es donde entra en juego XML (eXtensible Markup Language).

XML define un conjunto de reglas semánticas que permiten la organización de información de distintas maneras. Es un estándar definido por el W3C y ofrece muchas ventajas, entre ellas:

- ✓ Es un lenguaje bien formado. No puede haber etiquetas sin finalizar.
- ✓ Extensible. Permite ampliar el lenguaje mediante nuevas etiquetas y la definición de lenguajes nuevos
- ✓ Fácil de leer:
- ✓ Autodescriptivo. La estructura de la información de alguna manera está definida dentro del mismo documento
- ✓ Intercambiable. Portable entre distintas arquitecturas.
- ✓ Para su lectura e interpretación es necesario un **parser**, y hay productos y versiones libres.

Debido fundamentalmente a estas ventajas y a su sencillez, **el estándar XML ha sido ampliamente aceptado y adoptado para el almacenamiento e intercambio de información**, y como consecuencia de este uso se ha creado la necesidad de almacenar dicha información.

Y ¿cómo se almacena la información en formato XML de cara a su recuperación y consulta? Existen varias aproximaciones para organizar y almacenar información XML que fundamentalmente van a depender de los diferentes archivos o documentos XML que nos podemos encontrar (centrados en datos y centrados en texto).

Pero en definitiva, lo que se busca es una estrategia que permita almacenar y recuperar datos poco estructurados con la eficiencia de las Bases de Datos convencionales, y es por ello que surgen las Bases de Datos XML.

1.1.- Documentos XML centrados en datos y centrados en texto.

Como te hemos comentado antes, **los tipos de documentos XML que nos podemos encontrar** son:

- ✓ **Documentos centrados en datos**, con las siguientes características:
 - ✓ Muchos elementos de datos de pequeño tamaño.
 - ✓ Con estructura regular y bien definida.
 - ✓ Datos muy estructurados o semi-estructurados.
 - ✓ Dirigidos a utilización automática (por máquinas).
 - ✓ Ejemplos: Facturas, Pedidos Ficha de alumno.
- ✓ **Documentos centrados en texto, contenido o documentos** con las siguientes características:
 - ✓ Pocos elementos.
 - ✓ Con grandes cantidades de texto.
 - ✓ Con estructuras impredecibles en tamaño y contenido.
 - ✓ Datos poco estructurados.
 - ✓ Orientados a ser interpretadas por humanos.
 - ✓ Enfocados a sistemas documentales y de gestión de contenidos.
 - ✓ Ejemplos: Libros, Informes, Memorias, Artículos bibliográficos.

En la siguiente imagen ampliable puedes apreciar las características indicadas para estos dos tipos de documentos.



[\(link: AD06_CONT_R03_DatosTexto.jpg.\)](#)

En el siguiente enlace tienes un ejemplo de cada tipo de documento.

[Ejemplo de documento centrado en datos y documento centrado en texto. \(link: AD06_CONT_R04_DocumentoDatosTexto.odt \)](#) (0.02 MB)

Para saber más

Te proponemos el siguiente enlace para repasar el significado de documento XML bien formado.

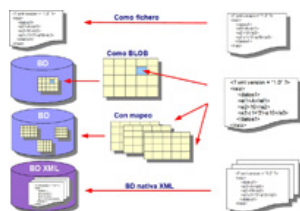
Documento XML bien formado. [\(link: http://flanagan.ugr.es/xml/documento.htm \)](http://flanagan.ugr.es/xml/documento.htm)

1.2.- Opciones de almacenamiento.

Para almacenar documentos XML tenemos las siguientes opciones:

- ✓ **Almacenamiento directo del fichero.** Es una opción pobre ya que las operaciones que podemos hacer sobre ellos son limitadas, las que proporcione el sistema de archivos.
- ✓ **Almacenar el documento en una base de datos existente** (base de datos relacional, orientada a objetos u objeto-relacional). En este caso existen las siguientes posibilidades:
 - ✓ **Directamente como una columna tipo binario grande (BLOB) dentro de una tabla.** En este caso se almacena el documento intacto. Es una buena estrategia si el documento XML contiene contenido estático, que solo será modificado cuando se reemplaza el documento completo. Almacenar el documento completo en formato de texto es fácil de implementar dado que no se necesita **mapeo** o traducción, pero limita las consultas y búsquedas de contenido.
 - ✓ **Mediante mapeo basado en tablas, o basado en objetos.** En ambos casos hay que realizar una transformación para ajustarlo a la estructura de la **BD** antes de almacenarlo, y normalmente esto conlleva prescindir de cierta información, que supondrá que el documento y su formato no se almacena y recupera de manera intacta. Además, en el caso de documentos centrados en texto, no podemos controlar todas las posibles estructuras del documento, y por tanto realizar un mapeo sobre la base de datos será prácticamente imposible.
- ✓ **Almacenar el documento en una base de datos nativa XML.** El documento, tanto si es centrado en datos como en texto, se almacena y recupera de forma intacta. Como veremos, será la mejor opción, sobre todo si el documento está basado en texto.

La siguiente imagen ampliable ilustra las anteriores opciones de almacenamiento:



[\(link: AD06_CONT_R05 almacenarXML.jpg.\)](#)

En la actualidad, cada vez más, las BD convencionales ofrecen posibilidades de almacenamiento XML, por lo que se habla de BD XML-compatible o BD XML-enabled. Por tanto, podemos hablar de dos **tipos de Sistemas de Bases de Datos que soportan documentos XML**:

- ✓ **BD XML-compatibles:** desglosan un documento XML en su correspondiente modelo relacional o de objetos.
- ✓ **BD XML Nativas:** respetan la estructura del documento, permiten hacer consultas sobre dicha estructura y recuperan el documento tal y como fue insertado originalmente

Las principales diferencias entre ambos tipos de BD XML son las siguientes:

- ✓ Una BD XML nativa proporciona un modelo de datos propio, mientras que un BD XML-compatible tiene ya un modelo de datos y añade una capa de software que permite de alguna manera almacenar documentos XML y recuperar los datos generando nuevos documentos XML.
- ✓ Una BD XML nativa debe manejar todos los tipos de documentos posibles, mientras que una BD XML-compatible solo puede manejar y almacenar los documentos que encajan dentro del modelo definido para ellos.

Para saber más

No dejes de visitar la página de Rounald Bourret, uno de los más importantes investigadores y consultores sobre BD XML. Aunque en inglés, en esta página encontrarás mucha información sobre el desarrollo de las bases de datos XML y sus especificaciones.

Página de Rounald Bourret sobre BD XML. [. \(link: http://www.rpbouret.com/xml/XMLAndDatabases.htm \)](http://www.rpbouret.com/xml/XMLAndDatabases.htm)

Caso práctico

Esta mañana, **María** ha estado hablando con una amiga del gremio. Su amiga lleva varios años trabajando en proyectos con tecnologías XML y además ha asistido personalmente a algunas conferencias del mismísimo Ronald Bourret, por lo que está más que al tanto de lo último en estas tecnologías.

María le ha pedido consejo sobre algunas de las bases de datos nativas XML, actualmente en el mercado.



Las **Bases de Datos nativas XML NXD** (Native XML Database) aunque existen desde hace años, en la actualidad siguen evolucionando, por lo que existen ciertas diferencias entre los productos de este tipo que podemos encontrar en el mercado. Lo que sí está bastante claro, es que **una NXD o BD XML debe cumplir las siguientes propiedades:**



- ✓ **Define un modelo lógico de datos XML**, estableciendo los elementos que son significativos, de forma que el documento XML se pueda almacenar y recuperar de manera intacta, esto es, con todos sus componentes. Por tanto, el modelo, como mínimo, debe tener en cuenta: los elementos, atributos, texto, secciones `CDATA`, y preservar el orden en el documento.
- ✓ **El documento XML es la unidad lógica de almacenamiento**, esto es, la unidad mínima de almacenamiento.
- ✓ **No tiene ningún modelo de almacenamiento físico subyacente concreto**. Pueden ser construidas sobre bases de datos relacionales, jerárquicas, orientadas a objetos o bien mediante formatos de almacenamiento propietarios.

Lo que realmente cambia en estas BD respecto a las convencionales, es el formato que soportan, ya que, están especializadas en almacenar documentos XML, almacenarlos y recuperarlos con todos sus componentes.

Otras características de las BD Nativas XML son las siguientes:

- ✓ **Documentos y colecciones**. Los documentos se pueden agrupar en unidades denominadas colecciones.
- ✓ **Indexación**. Permiten la creación de índices para acelerar las consultas realizadas frecuentemente.
- ✓ **Identificador único**. A cada documento XML se le asocia un identificador único, por el que será reconocido dentro del repositorio.
- ✓ **Consultas**. Soportan uno o más lenguajes de consulta. Entre ellos, uno de los más populares es XQuery (XML Query)
- ✓ **Actualizaciones**. Poseen diferentes estrategias para actualizar documentos, entre ellas la más popular es Update XQuery.
- ✓ **Validación**. No siempre se realiza validación de documentos, esto es, no necesitan un **DTD** o un XML Schema (W3C) para almacenar documentos, basta con que los documentos sean XML bien formados.
- ✓ Soportan **transacciones**, accesos concurrentes, control de accesos y backup como cualquier otro sistema de bases de datos.

Y ¿en qué situaciones puede resultar imprescindible su uso? Fundamentalmente en las siguientes situaciones:

- ✓ Existencia de documentos con anidamientos profundos.
- ✓ Importancia de preservar la integridad de los documentos.
- ✓ Frecuentes búsquedas de contenido.

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

Una BD XML nativa solo puede almacenar documentos XML válidos.

☐ Verdadero ☐ Falso

Para saber más

Consulta el siguiente enlace para recordar o bien familiarizarte con lo que son los DTDs y los esquemas XML.

Los DTDs y esquemas XML. (*link: <http://www.hipertexto.info/documentos/dtds.htm>)*

2.1.- Estrategias de almacenamiento.

Podemos diferenciar o clasificar las BD XML nativas en función del tipo de almacenamiento que utilicen, que puede ser:

- ✓ Almacenamiento basado en texto
- ✓ Almacenamiento basado en el modelo
- ✓ Soluciones desarrolladas específicamente para la gestión de documentos XML



¿En qué consiste el almacenamiento basado en texto y basado en modelo?

- ✓ El **almacenamiento basado en texto** consiste en almacenar el documento XML entero en forma de texto (fichero de texto), y proporcionar alguna funcionalidad de base de datos para acceder a él.

Se suelen aplicar técnicas de compresión para reducir el espacio de almacenamiento, utilizar índices adicionales para mejorar el acceso a la información, y se pueden definir sobre BD tradicionales o sistemas de ficheros. Básicamente existen dos posibilidades:

- ✓ Almacenar el documento como un binario largo (BLOB) en una base de datos relacional, o mediante un fichero, y proporcionar algunos índices sobre el documento que aceleren el acceso a la información.
 - ✓ Almacenar el documento en un almacén adecuado con índices, soporte para transacciones, etc.
-
- ✓ El **almacenamiento basado en modelo** consiste en definir un modelo de datos lógico, como DOM, para la estructura jerárquica de los documentos XML y almacenar el modelo binario del documento en un almacén existente o bien específico. En esta caso las posibilidades que existen son:
 - ✓ Traducir el DOM a tablas relacionales como elementos, atributos, entidades, etc.
 - ✓ Traducir el DOM a objetos en una BDOO.
 - ✓ Utilizar un almacén creado especialmente para esta finalidad

A continuación, te indicamos algunos ejemplos de BD XML nativas clasificadas según su sistema de almacenamiento:

- ✓ Sistema propietario: XIndex, Virtuoso, Tamino XML Server.
- ✓ Sistema relacional: eXist, DBCOM, XDB
- ✓ Sistema orientado a objetos: Ozone, MindSuite XDB.

¿Y qué ventajas proporcionan las BD XML nativas sobre otros sistemas de almacenamiento? Las **principales ventajas** son las siguientes:

- ✓ No necesitan mapeos adicionales.
- ✓ Conservan la integridad de los documentos.
- ✓ Permiten almacenar documentos heterogéneos.

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

El almacenamiento basado en modelo supone almacenar el documento como texto en un almacén adecuado con índices y que soporte transacciones.

☐ Verdadero ☐ Falso

Para saber más

Para refrescar tus conocimientos sobre el modelo de objetos de documentos DOM te recomendamos que visites el siguiente enlace. También obtendrás información sobre el parser SAX (Simple API for XML):

Los modelos DOM. (*link: <http://www.hipertexto.info/documentos/dom.htm>*)

2.2.- Colecciones y documentos.

En general, una BD XML tiene una estructura jerárquica organizada en colecciones y documentos XML. La estructura jerárquica comienza con un **nodo raíz ('/')**, del que parten colecciones y documentos.

Una colección:

- ✓ Es un conjunto de documentos agrupados, normalmente, en función de la información que contienen.
- ✓ Puede contener otras colecciones.



Un documento:

- ✓ Información XML
- ✓ Información de otro tipo y entonces se le denomina non-XML data (Datos no-XML)

[./link: AD06_CONT_R08_Colec_Docum.jpg./](#)

Comparando con una base de datos relacional o un sistema de archivos:

- ✓ Las colecciones juegan en las bases de datos nativas el papel de las tablas en las DB relacionales, o de un directorio en un sistemas de archivos.
- ✓ Los documentos juegan el papel de las filas de una tabla de una BD relacional o un fichero en un sistema de archivos.

Aunque depende de cada implementación, en muchos casos:

- ✓ Cada colección puede tener más de un DOCTYPE asociado.
- ✓ El elemento raíz del documento XML define a que DOCTYPE estará asociado el documento, en caso de no poseer ninguno, éste se crea dinámicamente. Esto posibilita el almacenamiento de documentos sin formato definido.
- ✓ La colección también puede tener asociado un Schema con información tanto física como lógica de la colección. La parte lógica define las relaciones y propiedades de los documentos XML y la física contiene información sobre el almacenamiento e indexación de los mismos.
- ✓ También se pueden almacenar documentos no-XML, para estos existe un DOCTYPE especial llamado non-XML.

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

Una colección solo puede almacenar documentos XML.

☐ Verdadero ☐ Falso

2.3.- Gestores nativos XML comerciales y libres

Te indicamos a continuación algunos ejemplos de gestores nativos XML clasificados según sean de código libre o código propietario. Los comerciales, generalmente ofrecen versiones de prueba con limitaciones de tiempo de uso y de funcionalidades. Los libres, a veces también ofrecen dualidad de licencia, comercial y libre.



Algunos de estos gestores nativos XML son los siguientes:

- ✓ **Gestores nativos XML comerciales** o de código propietario son:
 - ✓ **TaminoXML Server.** Es un gestor nativo XML de la empresa SoftwareAG. Es un producto comercial de alto rendimiento y disponibilidad, además de ser uno de los primeros SGBD XML nativos disponibles. Algunas de sus características son las siguientes:
 - ✓ Los documentos se almacenan en una base de datos propia y no se transforman en otro modelo.
 - ✓ Existe un espacio separado para documentos y para índices.
 - ✓ Soporta el lenguaje de consultas XQuery y APIs para Java, C, y .NET, entre otras.
- ✓ **TEXTML** de Isiasoft. Los documentos se almacenan en su formato nativo, sin ser mapeados.
 - ✓ Permite almacenar documentos sin DTD o esquema.
 - ✓ Proporciona índices de acuerdo a la propia estructura del documentos.
 - ✓ Permite la utilización de varios índices al mismo tiempo.
 - ✓ Incluye API para Java, WebDAV, OLE DB y .NET.
- ✓ **Gestores nativos XML libres** o de código abierto son:
 - ✓ **Qizx.** Es una base de datos nativa XML escrita en Java. Almacena los datos utilizando una representación que se basa en el modelo de XPath 2.0.
 - ✓ La representación utiliza compresión de manera que un documento y sus índices suelen utilizar menos espacio que el original XML.
 - ✓ Los documentos se almacenan en una jerarquía de colecciones.
 - ✓ Permite almacenar documentos sin DTD o esquema.
 - ✓ Soporta el lenguaje de consulta XQuery y sus extensiones, como XUpdate, así como API para Java.
 - ✓ **eXist.** Utiliza un sistema de almacenamiento propio (árboles B+ y archivos paginados). Se puede ejecutar como un servidor de base de datos independiente, como una biblioteca de Java embebida, o en el motor servlet de una aplicación Web.
 - ✓ Los documentos se almacenan en una jerarquía de colecciones.
 - ✓ Permite almacenar documentos sin DTD o esquema.
 - ✓ Soporta el lenguaje de consulta XQuery y sus extensiones, como XUpdate, así como API para Java.

Para saber más

Puedes consultar una relación extensa de bases de datos nativas XML, comerciales y libres, en el siguiente enlace:

Relación de BD XML nativas, comerciales y libres. (*link: <http://www.rpbourret.com/xml/XMLDatabaseProds.htm#native>*)

Si no sabes lo que es un árbol B+, consulta este enlace para conocer estas estructuras de datos:

Información sobre árboles B+ (*link: <http://es.wikipedia.org/wiki/%C3%81rbol-B%2B>*)

Caso práctico

Juan y María están muy satisfechos con **Ana**, realmente les está ayudando bastante con sus conocimientos de XML. Además, justo el gestor XML nativo que propuso **Ana** desde el principio, es uno de los que a **María** le sugirió su amiga. **Juan** le pregunta a **Ana** —¿Has trabajado antes con el gestor eXist-db? **Ana** le contesta —Pues....., sí. En clase, hicimos algunas prácticas con esta BD XML y realmente me gustó, es sencilla y está optimizada para consultas. Además tiene una versión libre. A lo que **Juan** le dice —Pues ahora es tu momento, serás nuestra profesora estos primeros días, para familiarizarnos con QeXist-db.



eXist es un SGBD libre de código abierto que almacena datos XML de acuerdo a un modelo de datos XML. El motor de base de datos está completamente escrito en Java, soporta los estándares de consulta XPath, XQuery y XSLT. Con el SGBD se dan aplicaciones que permiten ejecutar consultas directamente sobre la BD.



Los documentos XML se almacenan en colecciones, desde un punto de vista práctico el almacén de datos funciona como un sistema de ficheros. Cada documento está en una colección. Dentro de una colección pueden almacenarse documentos de cualquier tipo.

En la carpeta eXist\webapp\WEB-INF\data es donde se guardan los ficheros más importantes de la BD:

- ✓ **dom.dbx**, el almacén central nativo de datos; es un fichero paginado donde se almacenan todos los nodos del documento de acuerdo al modelo DOM del W3C.
- ✓ **collections.dbx**, se almacena la jerarquía de colecciones y relaciona esta con los documentos que contiene; se asigna un identificador único a cada documento de la colección que es almacenado también junto al índice.
- ✓ **elements.dbx**, es donde se guarda el índice de elementos y atributos. El gestor automáticamente indexa todos los documentos utilizando índices numéricos para identificar los nodos del mismo (elementos, atributos, texto y comentarios).
- ✓ **words.dbx**, por defecto eXist indexa todos los nodos de texto y valores de atributos dividiendo el texto en palabras. En este fichero se almacena esta información. Cada entrada del índice está formada por un par <id de colección, palabra> y después una lista al nodo que contiene dicha palabra.

3.1.- Instalación de eXist-db

eXist puede funcionar de distintos modos:

- Funcionando como servidor.
- Insertado dentro de una aplicación Java.
- En un servidor J2EE.

En el sitio <http://exist-db.org> *(link: <http://exist-db.org/exist/apps/homepage/index.html>)* podremos descargar la última versión de la BD.

4.- El lenguaje de consultas XQuery.

Caso práctico

Ana está orgullosa en su nueva faceta de profesora. **Juan y María** la han felicitado y están satisfechos con la introducción que les ha dado sobre eXist-db. **Ana** les dice —Antes de empezar con la aplicación en Java, necesitamos familiarizarnos con el lenguaje de consultas XQuery.

A lo que **Juan** le pregunta —¿Conoces algo del lenguaje?"

Ana sonríe y contesta —Pues...sí. En clase vimos también algunos ejemplos sencillos de uso.

Juan, con cara de asombro dice —Está claro que el nuevo Ciclo Formativo de **DAM** incorpora muchos temas punteros que yo no domino. El próximo curso iniciaré sin falta el estudio de algunos módulos.



XQuery es un lenguaje de consulta diseñado para extraer información de colecciones de datos expresadas en XML. Entre las **principales características de XQuery** vamos a destacar las siguientes:

- ✓ Está **basado en el lenguaje XPath**, (XML Path Language), y se fundamenta en él para realizar la selección de información y la iteración a través del conjunto de datos XML.
- ✓ Es un **lenguaje declarativo**, lo que significa que, en vez de ejecutar una lista de comandos como un **lenguaje procedimental** clásico, cada consulta es una expresión que es evaluada y devuelve un resultado, al igual que en **SQL**.



Podemos decir que **XQuery es a XML lo mismo que SQL es a las bases de datos relacionales**. Sin embargo, aunque XQuery y SQL puedan considerarse similares, el modelo de datos sobre el que se sustenta XQuery es muy distinto del modelo de datos relacional sobre el que sustenta SQL, ya que XML incluye conceptos como jerarquía y orden de los datos que no están presentes en el modelo relacional.

Por ejemplo, a diferencia de SQL, en XQuery es importante y determinante el orden en que se encuentren los datos, ya que no es lo mismo buscar una etiqueta nombre dentro de una etiqueta autor que todas las etiquetas nombre del documento (que pueden estar anidadas dentro de una etiqueta autor o fuera).

Los **principales tipos de expresiones de XQuery** son:

- ✓ Expresiones XPath, para navegar por los documentos.
- ✓ Expresiones FLWOR (For, Let, Where, Order, Return) para iterar por los elementos de un conjunto de datos.

Pero XQuery también admite:

- ✓ Constructores para generar nodos y contenido dinámico.
- ✓ Condicionales (IF, THEN ELSE) para construir el resultado en base a alguna condición.
- ✓ Cuantificadores (SOME, ANY) para chequear la existencia de algún elemento que cumpla una condición.
- ✓ Listas a las que se pueden aplicar operadores (UNION,...) y funciones de agregación (AVG, COUNT,...).

Para saber más

En este enlace puedes profundizar sobre el lenguaje XQuery.

Especificación completa de XQuery. ([link: http://www.w3.org/TR/XQuery/](http://www.w3.org/TR/XQuery/))

El siguiente enlace te proporciona a su vez enlaces muchos enlaces interesantes a varias tecnologías asociadas a XQuery, así como a XQuery.

Tecnologías asociadas con XQuery. ([link: http://gim.unex.es/blogs/ljarevalo/tag/XQuery-xml-xslt-XPath/](http://gim.unex.es/blogs/ljarevalo/tag/XQuery-xml-xslt-XPath/))

4.1.- Modelo de datos.

El modelo de datos en que se sustenta XQuery es el modelo de datos de XPath.

XPath modela un documento XML como una estructura jerárquica en forma de árbol. El árbol está **formado por nodos, y hay siete tipos de nodos**: raíz, elemento, texto, atributo, espacio de nombres, instrucción de procesamiento y comentario.



[\(link: AD06_CONT_R17_ArbolXPath.jpg.\)](#)

Los **principales nodos** de la estructura jerárquica o en árbol en un documento XML son: (puedes verlos en la imagen ampliable superior)

- ✓ **Nodo raíz o /**. Es el primer nodo del documento.
- ✓ **Nodo elemento**. Cualquier elemento de un documento XML. Cada nodo elemento posee un padre y puede o no tener hijos. En el caso de que no tenga hijos, es un nodo hoja.
- ✓ **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. Esto sería un nodo atributo.

Debes conocer

En el siguiente enlace tienes una ilustración gráfica más detallada del paso de un documento XML a un árbol de nodos XML, así como la explicación en detalle de cada tipo de nodo. (Apartado 2.Modelo de datos XPath)

Nodos de un árbol XML. ([link: http://geneura.ugr.es/~victor/cursillos/xml/XPath/](http://geneura.ugr.es/~victor/cursillos/xml/XPath/))

Autoevaluación

Señala la opción correcta. Entre los nodos de la estructura jerárquica de un árbol XML están:

- ☐ ([link:](#))
Nodo etiqueta y nodo atributo.
- ☐ ([link:](#))
Nodo padre.
- ☐ ([link:](#))
Nodo hijo.
- ☐ ([link:](#))
Nodo raíz y nodo elemento.

4.2.- Caminos de localización.

¿Cómo se localiza cada uno de esos nodos en el árbol XML? Mediante una expresión XPath conocida como camino o ruta de localización. **Un camino de localización:**

- ✓ Selecciona un conjunto de nodos relativo al nodo de contexto.
- ✓ Puede contener recursivamente expresiones utilizadas para filtrar conjuntos de nodos.
- ✓ Al ser evaluado, devuelve el conjunto de nodos seleccionados por el camino de localización.
- ✓ Se construye siguiendo unas reglas de sintaxis y semántica.



Hay dos **tipos de caminos de localización:**

- ✓ **Caminos relativos.** Son una secuencia de uno o más pasos de localización separados por **/**.
 - ✓ Los pasos se componen de izquierda a derecha.
- ✓ **Caminos absolutos.** Consiste en **/** seguido, opcionalmente, por un camino de localización relativo.
 - ✓ Una **/** por sí misma selecciona el nodo raíz del documento que contiene al nodo contextual.

Los siguientes, son **algunos ejemplos de caminos de localización:**

- ✓ **cuadro** selecciona los elementos cuadro hijos del nodo contextual.
- ✓ **cuadro//titulo** selecciona los elementos titulo descendientes de los elementos cuadro hijos del nodo contextual.
- ✓ ***** selecciona todos los elementos hijos del nodo contextual.
- ✓ **@año** selecciona el atributo año del nodo contextual
- ✓ **@*** selecciona todos los atributos del nodo contextual
- ✓ **cuadro[1]** selecciona el primer hijo cuadro del nodo contextual
- ✓ **cuadro[@año=1907]** selecciona todos los hijos cuadro del nodo contextual que tengan un atributo año con valor 1907.

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

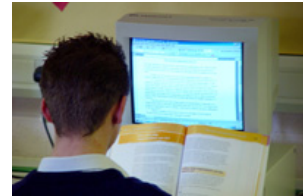
El camino: //curso selecciona los elementos curso descendientes del nodo contextual.

☐ Verdadero ☐ Falso

4.3.- Primeras consultas XQuery.

Una **consulta XQuery** es una expresión que lee una secuencia de datos en XML y devuelve como resultado otra secuencia de datos en XML, donde:

- ✓ Una **secuencia** es un conjunto ordenado de cero o más ítems.
- ✓ Un **ítem** es cualquier tipo de nodo del árbol XML o un **valor atómico**.



Las funciones que se pueden invocar para **referirnos a colecciones y documentos dentro de la BD** son las siguientes:

- ✓ `collection(camino de la colección)`
- ✓ `doc(camino del documento)`

Así por ejemplo:

- ✓ La consulta `collection(/Books)`: devuelve el contenido de la colección de ruta absoluta `/Books`.
- ✓ La consulta `doc(/Empresa.xml)`: devuelve el documento `/Empresa.xml` completo.

Otros **ejemplos de consultas XQuery basadas en expresiones XPath** son los siguientes:

- ✓ La consulta `collection(/Books)//book/title` devuelve los nodos `title` de todos los libros (`book`) de la colección `/Books`
- ✓ Si se utilizan espacios de nombres o namespaces, entonces la consulta anterior se redactaría de la siguiente forma: declare namespace t = `http://www.qizx.com/namespace/Tutorial`; `collection(/Books)//t:book/t:title`
- ✓ La consulta `doc(/Empresa.xml)//nombre` devuelve todos los nodos `nombre` del documento `/Empresa.xml`

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

La **expresión** `colletcion("/Cursos")//curso[aula=2]/profesor` **devuelve los cursos y los profesores con cursos en el aula 2.**

☐ Verdadero ☐ Falso

Para saber más

Te recomendamos que visites el siguiente enlace para profundizar en el concepto y uso de los namespaces.

Espacios de nombres o namespaces en XML. (*link: <http://flanagan.ugr.es/xml/nombres.htm>)*

4.4.- Expresiones FLWOR.

Los ejemplos de consultas XQuery vistos en el apartado anterior, son la manera más sencilla de realizar búsquedas y selecciones de nodos concretos en una BD XML. Pero existe otra manera mucho más potente de realizar este trabajo, mediante lo que se denomina consultas o **expresiones FLWOR** (leído como flower), siendo FLWOR las siglas de For, Let, Where, Order y Return.

Se trata de una **expresión que permite la unión de variables sobre conjuntos de nodos y la iteración sobre el resultado**. Las diferentes **cláusulas de una expresión FLWOR** son:

- ✓ **For**. Permite seleccionar los nodos que se quieren consultar, guardándose su valor en una variable (identificador que comienza por \$). Al conjunto de valores de la variable se le llama **tupla**.
- ✓ **Let**. (opcional). Asocia valores a variables.
- ✓ **Where** (opcional). Permite filtrar los resultados según una condición.
- ✓ **Order** (opcional). Permite ordenar la secuencia de valores o resultados.
- ✓ **Return**. Genera los valores de salida o devueltos.

En la siguiente imagen, puedes ver un ejemplo de una consulta (izquierda) donde aparecen las 5 cláusulas. La consulta devuelve los nombres de los cursos con 20 plazas, ordenados por nombre de curso (derecha).



[\(link: AD06_CONT_R21_QizxStudioFLWOR.jpg.\)](#)

Una expresión FLWOR vincula variables a valores con las cláusulas for y let y utiliza esos vínculos para crear nuevas estructuras de datos XML.

A continuación se muestra otro ejemplo de consulta XQuery. La siguiente consulta (izquierda) devuelve los nombres de los cursos con cuota mensual (derecha). Como cuota es un atributo del elemento precio, recuerda que se le antecede con un carácter @.



[\(link: AD06_CONT_R22_QizxStudioFLWOR2.jpg.\)](#)

Para saber más

En el siguiente enlace tienes muchos ejemplos de consultas XQuery con expresiones FLWOR.

Ejemplos XQuery con expresiones FLWOR ([link: http://www.stylusstudio.com/XQuery_flwor.html](http://www.stylusstudio.com/XQuery_flwor.html))

4.5.- XQuery Update Facility.

El lenguaje **XQuery solo proporciona expresiones para la realización de consultas sobre documentos XML**, pero **no su actualización (inserción, modificación o eliminación de nodos)**.

XQuery Update Facility es una extensión de XQuery que permite la actualización de documentos mediante las cláusulas `insert`, `delete`, `replace` y `rename`.

El **funcionamiento de las cláusulas de XQuery Update** es el siguiente:

- ✓ **insert**. Permite la inserción de uno o varios nodos antes (before) o después (after) del nodo indicado. También se puede insertar al principio (as `first` into) o al final del documento (as `last` into).



[\(link: AD06_CONT_R24_UpdateXQueryL.jpg.\)](#)

- ✓ **delete**. Elimina uno o varios nodos del documento.

- ✓ **replace**. Tiene dos funciones:

- ✓ Modificar el valor del nodo
- ✓ Modificar el nodo completo

- ✓ **rename**. Renombra un nodo (elemento, atributo o instrucciones de proceso) sin afectar a su contenido.

A continuación te mostramos dos ejemplos sencillos, uno de `insert` y otro de `delete`:

- ✓ Eliminar la empresa de `id=2` en el documento `Empresa.xml`: `delete node doc(Empresa.xml)//empresa[@id=2]`
- ✓ Insertar el nodo `tipoAccesible/tipo` al final del documento `/Aulas/aula3`: `insert node tipoAccesible/tipo as last into doc(Aulas/aula3.xml)//aula`

En el siguiente enlace puedes otros ejemplos de actualizaciones usando XQuery Update:

[Ejemplos de XQuery Update \(link: http://xqilla.sourceforge.net/XQueryUpdate.\)](http://xqilla.sourceforge.net/XQueryUpdate/)

Autoevaluación

Señala si la siguiente afirmación es verdadera o falsa.

Mediante `rename` se puede sustituir el valor de un nodo.

☐ Verdadero ☐ Falso

Para saber más

En el siguiente enlace puedes ver un documento tipo resumen con muchos de los aspectos tratados en esta unidad hasta el momento.

XML y Bases de Datos. [\(link: http://www.di.uniovi.es/~labra/cursos/ext08/pres/bdxml.pdf\)](http://www.di.uniovi.es/~labra/cursos/ext08/pres/bdxml.pdf)

5.- Trabajar con colecciones y documentos desde Java.

Caso práctico

—¡Ha llegado la hora de manejar la base de datos desde Java! —exclama **Ana**, y añade— ¿queréis que os indique cómo crear colecciones y añadir documentos?

Juan contesta —Pues claro **Ana**, ¿sabes cual es el API principal que proporciona este gestor para trabajar con Java?

Ana contesta —De memoria no lo recuerdo, pero sí recuerdo que la propia distribución dispone de datos de prueba que podemos utilizar en nuestros primeros ejemplos.

María, que escucha con atención dice —¡Perfecto!, vamos a ello.



En este apartado vamos a ver distintas APIs que acceden a la BD eXist para procesar documentos XML.

Actualmente, la API xmldb está siendo utilizada por la mayoría de bases de datos. La API propuesta se basa en tres paquetes: **org.xmldb.api**, **org.xmldb.api.base** y **org.xmldb.api.modules**.

Los componentes básicos empleados por el código XML: DB API son los drivers, las colecciones, los recursos y los servicios. Veamos cada uno de ellos:

- ✓ **Los drivers** son implementaciones de la interfaz que encapsula la lógica de acceso a la base de datos XML. Los proporciona el proveedor del producto y debe ser registrado con el gestor de base de datos. Ejemplo:

```
String driver = "org.exist.xmlldb.DatabasImpl"; //Driver para eXist
Class c1 = Class.forName(driver); //Cargar del driver
Database database = (Database) c1.newInstance(); //Instancia de la BD
DatabaseManager.registerDatabase(database); //Registro del driver
```

- ✓ **Una colección** es un contenedor de recursos y otras sub-colecciones. La API define dos recursos diferentes: XMLResource y BinaryResource. Un XMLResource representa un documento XML o un fragmento del documento, seleccionados por la ejecución de una consulta XPath. Una vez utilizado el recurso se debe de cerrar. Ejemplo:

```
String URI="xmldb:exist://localhost:8080/exist/xmlrpc/db/Pruebas"; //Colección
String usu="admin"; //Usuario
String usuPwd="admin"; //Clave
Collection col = DatabaseManager.getCollection(URI, usu, usuPwd);
```

- ✓ **Los servicios** se solicitan para tareas como consultar una colección con XPath o la gestión de una colección. Ejemplo:

```
XPathQueryService servicio = (XPathQueryService) col.getService("XPathQueryService", "1.0");
ResourceSet result = servicio.query("for $em in /EMPLEADOS/EMP_ROW return $em");
```

Para ejecutar la consulta llamaremos al método **nombre_servicio.query(xpath)**, este método devuelve un **ResourceSet**, que contiene el recurso, en el ejemplo anterior el resultado de la consulta es devuelto en result. El siguiente código lo escribiremos para recorrer el recurso result devuelto por la consulta anterior:

```
ResourceIterator i; //se utiliza para recorrer un set de recursos
i = result.getIterator();
if (!i.hasMoreResources ()) {
    System.out.println(" LA CONSULTA NO DEVUELVE NADA.");
}
while (i.hasMoreResources ()) {
    Resource r = i.nextResource();
    System.out.println((String) r.getContent());
}
```

Donde **result.getIterator()** nos da un iterador sobre el recurso, cada recurso contiene el valor seleccionado por la expresión XPath. El método **getContent()**, devuelve el contenido del recurso, en nuestro ejemplo devuelve la consulta y el tipo es String. Para localizar si existe un documento en una colección utilizaremos este código:

```
Collection col= DatabaseManager.getCollection(URI, usu, usuPwd);
XMLResource res = null;
res = (XMLResource)col.getResource("empleados.xml");
if(res == null)
    System.out.println("NO EXISTE EL DOCUMENTO");
```

5.1.- Librerías y ejemplo de conexión

Para realizar un programa que consulte la BD eXist debemos incluir las siguientes librerías: *exist.jar*, *exist-optional.jar*, *xmlldb.jar*, *xml-api-13.04.jar*, *xmlrpc-client-3.1.1.jar*, *xmlrpc-common-3.1.1.jar* situadas en la instalación de eXist y *log4j-1.2.15.jar*.

En el siguiente ejemplo vemos cómo utilizar las clases vistas anteriormente, este ejercicio realiza una conexión con la BD para acceder al documento *empleados.xml* y obtener en XML los empleados del departamento 10.

```
import org.xmlldb.api.*;
import org.xmlldb.api.base.*;
import org.xmlldb.api.modules.*;

public static void verempleados10(String[] args) throws XMLDBException {
    String driver = "org.exist.xmlldb.DatasourceImpl"; //Driver para eXist
    Collection col = null; // Colección
    String URI="xmlldb:exist://localhost:8080/exist/xmlrpc/db/Pruebas"; //URI colección
    String usu="admin"; //Usuario
    String usuPwd="admin"; //Clave
    try {
        Class cl = Class.forName(driver); //Cargar del driver
        Database database = (Database) cl.newInstance(); //Instancia de la BD
        DatabaseManager.registerDatabase(database); //Registro del driver
    } catch (Exception e) {
        System.out.println("Error al inicializar la BD eXist");
        e.printStackTrace();
    }
    col = DatabaseManager.getCollection(URI, usu, usuPwd);
    if(col == null)
        System.out.println(" *** LA COLECCION NO EXISTE. ***");
    XPathQueryService servicio = (XPathQueryService) col.getService("XPathQueryService", "1.0");
    ResourceSet result = servicio.query ("for $em in /EMPLEADOS/EMP_ROW[DEPT_NO=10] return $em");
    // recorrer los datos del recurso.
    ResourceIterator i;
    i = result.iterator();
    if (!i.hasMoreResources())
        System.out.println(" LA CONSULTA NO DEVUELVE NADA.");
    while (i.hasMoreResources()) {
        Resource r = i.nextResource();
        System.out.println((String) r.getContent());
    }
    col.close(); //borramos
} // FIN verempleados10
```


5.2.- Operaciones sobre colecciones y documentos

La API **XMLDB** nos va a permitir además de consultar documentos en la BD, crear y eliminar colecciones, y crear y eliminar documentos.

- ✓ **Crear una colección:** para crear una nueva colección, se llama al método ***createCollection*** del servicio ***CollectionManagementService***. El siguiente ejemplo crea la colección NUEVA_COLECCION dentro de la colección col:

```
CollectionManagementService mgtService =  
    (CollectionManagementService)col.getService("CollectionManagementService", "1.0");  
mgtService.createCollection("NUEVA_COLECCION");
```

- ✓ **Borrar una colección:** para borrar utilizamos el método ***removeCollection*** :

```
mgtService = (CollectionManagementService)col.getService("CollectionManagementService", "1.0");  
mgtService.removeCollection("NUEVA_COLECCION");
```

- ✓ **Crear un nuevo documento:** este ejemplo añade un documento nuevo a la colección ***col***, el documento se llama NUEVOS_DEP.xml, y se encuentra en nuestro disco, en la carpeta donde está el programa. Utilizamos el paquete ***java.io.File***, para declarar el fichero a subir a la BD, y el método ***createResource*** para crear el recurso.

```
import java.io.File;  
.....  
File archivo= new File("NUEVOS_DEP.xml");  
if(!archivo.canRead()) System.out.println("ERROR AL LEER EL FICHERO");  
else{  
    Resource nuevoRecurso = col.createResource(archivo.getName(), "XMLResource");  
    nuevoRecurso.setContent(archivo); //comprueba si es un archivo  
    col.storeResource(nuevoRecurso);  
};
```

- ✓ **Borrar un documento de la colección:** este ejemplo borra el documento creado anteriormente, comprueba si existe. Se utiliza el método ***removeResource***:

```
try {  
    Resource recursoParaBorrar = col.getResource("NUEVOS_DEP.xml"); col.removeResource(recursoPar  
}  
catch(NullPointerException e) {  
    System.out.println("No se puede borrar. No se encuentra.");  
}
```

Citas para pensar

"Saber que se sabe lo que se sabe y que no se sabe lo que no se sabe; he aquí el verdadero saber."

Confucio

Caso práctico

Esta mañana, **María, Juan y Ana** se han reunido con **Ada** para hablar de los avances del proyecto. Ada, ha quedado bastante satisfecha y les ha preguntado —¿Habéis considerado la posibilidad de establecer índices personalizados para optimizar las consultas?

María contesta —¡Claro **Ada**! Si mal no recuerdo, el personal de la academia de inglés nos insistió en que algunas búsquedas o consultas de información resultaban eternas.

Juan añade —Hoy vamos a tratar precisamente este tema.



Como cualquier otra BD XML, eXist-db utiliza índices para optimizar y aumentar la velocidad de las consultas realizadas a la BD.

De manera predeterminada, eXist-db indexa la mayor parte de la información disponible en documentos XML: elementos, atributos, nodos, y texto completo. Esto se realiza automáticamente, por lo tanto, en la mayoría de los casos no hay necesidad de especificar explícitamente nuevos índices.

En el siguiente enlace podrás ver los tipos de índices que soporta eXist-db y cómo se configuran:

[Configuración de índices \(link: <https://exist-db.org/exist/apps/doc/indexing.xml>\).](https://exist-db.org/exist/apps/doc/indexing.xml)



7.- Gestión de transacciones.

Caso práctico

Esta mañana, **Ada** le ha pedido a **Antonio** que pasara a ver la marcha del proyecto de **Ana, Juan y María**, a ver si se animaba a estudiar un CF. Ada sigue insistiendo en que su amigo debe estudiar un Ciclo Formativo de Informática. Al entrar a la sala de programación, **Antonio** ha escuchado decir a **Ana** —¿Habéis visto que eXist-db debe confirmar las transacciones después de cada actualización? —**Antonio** saluda y dice— Esa me la sé, ¿no son las transacciones lo que se gestiona con commit y rollback? —**Ana**, Juan y María sonríen y lo saludan.



En muchos de los ejemplos que has visto, ha estado presente el concepto de **transacción**: conjunto de sentencias que se ejecutan formando una unidad de trabajo, de forma indivisible, de manera que se ejecutan todas o no se ejecuta ninguna.

Mediante la gestión de transacciones, los gestores XML proporcionan un acceso concurrente a los datos almacenados, mantienen la integridad y seguridad de los datos, y proporcionan un mecanismo de recuperación de la base de datos ante fallos.



eXist-db soporta transacciones que cumplen el criterio **ACID**:

- ✓ **Atomicidad**: Se deben cumplir todas las operaciones de la transacción o no se cumple ninguna; no puede quedar a medias.
- ✓ **Consistencia**: La transacción solo termina si la base de datos queda en un estado consistente.
- ✓ **Isolation** (Aislamiento): Las transacciones sobre la misma información deben ser independientes, para que no interfieran sus operaciones y no se produzca ningún tipo de error.
- ✓ **Durabilidad**. Cuando la transacción termina el resultado de la misma perdura, y no se puede deshacer aunque falle el sistema.

7.1.- Los métodos commit() y rollback().

Los métodos para gestionar transacciones en eXist-db los proporciona son los siguientes:

- ✓ `commit()`. Confirma la transacción actual. Si este método se completa sin errores, las actualizaciones realizadas en la transacción garantizan la **persistencia**, y se hacen visibles a otras **sesiones**.
 - ✓ Una vez confirmada, desbloquea todos los objetos de la BD, previamente bloqueados.
 - ✓ Finalizada la transacción, el estado de la BD también refleja los cambios realizados posiblemente por otras transacciones.
- ✓ `rollback()`. Cancela la transacción actual.
 - ✓ Desbloquea todos los objetos de la BD, previamente bloqueados.
 - ✓ Después de un `rollback()` se refresca o actualiza el estado de la BD, de manera que se reflejan los cambios realizados por otras transacciones.

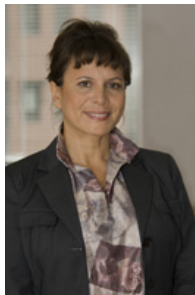


8.- Tratamiento de excepciones.

Caso práctico

Ada se ha vuelto a reunir con **María** y **Juan**. Les ha insistido en que, como siempre, no deben descuidar la robustez de la aplicación. Ya saben que los clientes son muy exigentes y que una aplicación que falla de manera abrupta y se bloquea, no es el sello de la empresa BK Programación.

Juan y **María**, al unísono le han contestado —Tranquila **Ada**, la aplicación va a capturar y procesar todas las posibles excepciones. No habrá error que se le resista.



Esta excepción se lanza cuando se produce un error en la API XML:DB. Contiene dos códigos de error, uno es el código de error XML de la API, y el otro es definido por el proveedor específico. El error del proveedor vendrá definido por *ErrorCodes.VENDO_ERROR*.










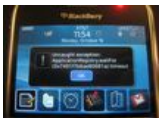
Cuando se produce un error con **XMLDBException** podemos acceder a cierta información usando el método *getMessage()* que devuelve una cadena que describe el error.

- El error **NullPointerException**, es el error que se dispara siempre que no se haya inicializado la BD, es el caso de escribir mal el driver.
- Si escribimos mal la URI, se disparará **XMLDBException** a la hora de asignar la colección.
- Si escribimos mal la carpeta donde se encuentra la colección, se disparan los **NullPointerException** siguientes, ya que el servicio no se ha podido crear.
- Si escribimos mal la query se dispara el error **XMLDBException**, a la hora de crear el service.

Cuando se produce un error con **XQException** disponemos de dos métodos que nos van a permitir saber la causa. Estos métodos son *getMessage()* que devuelve una cadena que describe el error y *getCause()* que nos indica la causa.

Anexo.- Licencias de recursos.

Licencias de recursos utilizados en la Unidad de Trabajo.

Recurso (1)	Datos del recurso (1)	Recurso (2)	Datos del recurso (2)
	<p>Autoría: Isabel M. Cruz Granados.</p> <p>Licencia: Uso Educativo-nc.</p> <p>Procedencia: Captura de pantalla del programa QizxStudio de Qizx Free Engine Edition.</p>		<p>Autoría: Isabel M. Cruz Granados.</p> <p>Licencia: Uso Educativo-nc.</p> <p>Procedencia: Captura de pantalla del programa QizxStudio de Qizx Free Engine Edition.</p>
	<p>Autoría: Isabel M. Cruz Granados.</p> <p>Licencia: Uso Educativo-nc.</p> <p>Procedencia: Captura de pantalla del programa QizxStudio de Qizx Free Engine Edition.</p>		<p>Autoría: Isabel M. Cruz Granados.</p> <p>Licencia: Uso Educativo-nc.</p> <p>Procedencia: Captura de pantalla del programa QizxStudio de Qizx Free Engine Edition.</p>
	<p>Autoría: Emma Gracia Lor.</p> <p>Licencia: CC-by-nc-sa.</p> <p>Procedencia: http://www.flickr.com/photos/paideiaeducacion/5102909303/</p>		<p>Autoría: fsse8info.</p> <p>Licencia: CC BY-SA.</p> <p>Procedencia: http://www.flickr.com/photos/fsse8info/4194980532/</p>
	<p>Autoría: Howard Stanbury.</p> <p>Licencia: CC BY-NC-SA.</p> <p>Procedencia: http://www.flickr.com/photos/stanbury/5852623652/</p>		<p>Autoría: Justin See.</p> <p>Licencia: CC BY.</p> <p>Procedencia: http://www.flickr.com/photos/koalazyr/</p>
	<p>Autoría: Adam Prince.</p> <p>Licencia: CC BY-NC-SA.</p> <p>Procedencia: http://www.flickr.com/photos/adam_prince/3908404628/</p>		<p>Autoría: Tantek Çelik.</p> <p>Licencia: CC BY-NC.</p> <p>Procedencia: http://www.flickr.com/photos/tantek/</p>

