Almacenamiento de la información.

Caso práctico

Antes de irse de vacaciones con su familia, **María y Félix tienen una reunión con el responsable del departamento de informática de la empresa, Juan**, en la que éste último les explica que aún se puede mejorar la gestión de datos de la empresa.

nate steiner - www.flickr.com (CC BY)

Juan opina que ya que en la empresa **se ha impuesto el uso de la tecnología XML**, sería mejor utilizar un sistema de almacenamiento y consulta de datos compatible con dicha tecnología.

Van a intentar utilizar un sistema de almacenamiento de datos, diferente de la base de datos relacional que han usado hasta ahora, que es compatible con la tecnología **XML** que están utilizando.

Además, para acceder a los datos de esta base de datos nativa van a utilizar un lenguaje de consultas llamado **XQuery**, que equivale al **SQL** usado con las bases de datos relacionales.

GOBIERNO DE ESPAÑA
MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Ministerio de Educación y Formación Profesional. (Dominio público)

1.- Utilización de XML para el almacenamiento de la información.

Caso práctico

Mientras **María** está de vacaciones con su familia le **explica a su marido, José Ramón**, las nuevas **modificaciones** que va a realizar Juan en el sistema informático de la empresa.

Comienza por explicarle lo que es un sistema de almacenamiento, su relación con la tecnología **XML** y los distintos ámbitos de aplicación que pueden tener.

Como ya hemos visto, **XML**, es un estándar potente y de amplia aceptación para **guardar y comunicar información acerca de objetos**. Permite la codificación de información, separada de la forma en la que se debe presentar al usuario. Cuando se desea encontrar un fragmento específico de información en los contenidos de un nodo o atributo **XML**, es necesario procesar completamente todo el archivo **XML**.

xadrian - www.flickr.com (CC BY-NC-SA)

Podemos pensar en **XML** como en una base de datos. Visto así, una **base de datos XML** se puede ver como una colección de **documentos XML**. Cada documento **XML** representa un registro de la base de datos; es un archivo en el sistema de archivos y contiene una cadena válida **XML**.

La estructura de un **documento XML** suele seguir un mismo esquema **XML**, aunque no es necesario que sea así. Este es uno de los beneficios de las bases de datos **XML**, ya que cada archivo se puede configurar de forma estructurada por lo que es independiente, pero fácilmente accesible.

La principal ventaja de usar bases de datos **XML** es que proporcionan una gran flexibilidad (gracias a tener colecciones de documentos con un esquema independiente), lo que conlleva facilidad a la hora de crear aplicaciones que usen a la Base de Datos **XML**.

Esta flexibilidad es un gran valor, especialmente en los últimos años, en los que se ha visto la necesidad de contar con estándares para el intercambio de información. Estos estándares ayudan a que las organizaciones puedan compartir su información de una manera más cómoda, automática y eficiente.

Autoevaluación

La tecnología XML:

- [ ] Permite tratar a los ficheros como si fuesen base de datos.

- [ ] Facilita el desarrollo de las aplicaciones al tener el esquema independiente de los datos.

- [ ] Hace que el compartir información sea más cómodo.

- [ ] Permite el acceso directo a los datos buscados, es decir sin recorrer todo el archivo.

Mostrar retroalimentación

Solución

1. Correcto
2. Correcto
3. Correcto
4. Incorrecto

1.1.- Ámbitos de aplicación.

Foreverdigital - www.flickr.com (CC BY-NC-ND)

Los documentos y los requerimientos de almacenamiento de datos **XML** pueden ser agrupados en dos categorías generales:

- Sistemas centrados en los datos. Cuando los documentos **XML** tienen una estructura bien definida y contienen datos que pueden ser actualizados y usados de diversos modos. Es apropiada para **`ítems`** como contenidos de periódicos, artículos, publicidad, facturas, órdenes de compra... y algunos documentos menos estructurados.
- Sistemas centrados en los documentos. Cuando los documentos tienden a ser más impredecibles en tamaño y contenido. Presentan más tipos de datos, de tamaño más variable, con reglas flexibles para campos opcionales y para el propio contenido.
  Los sistemas de almacenamiento **XML** deben acomodarse eficientemente con ambos tipos de requerimientos de datos, dado que **XML** está siendo usado en sistemas que administran ambos tipos de datos.

La mayoría de los productos se enfocan en servir uno de esos formatos de datos mejor que el otro. Las bases de datos relacionales tradicionales son mejores para tratar con requerimientos centrados en los datos, mientras que los sistemas de administración de contenido y de documentos, suelen ser mejores para almacenar datos centrados en el documento.

Los sistemas de bases de datos deben ser capaces de exponer los datos relacionales como un documento **XML** y almacenar un documento **XML** recibido como datos relacionales para transferir, obtener y almacenar los datos.

Autoevaluación

Elige las características que se ajusten a los sistemas centrados en datos:

- [ ] Contiene datos con una estructura no muy definida.
- [ ] Es apropiada para publicidad.
- [ ] Los datos pueden actualizarse.
- [ ] Contiene datos con una estructura muy definida.

Mostrar retroalimentación

Solución

1. Correcto
2. Correcto
3. Correcto
4. Incorrecto

2.- Sistemas de almacenamiento de la información.

Caso práctico

Mientras María disfruta de sus vacaciones **Félix** y **Juan** estudian si el cambio propuesto por este último es viable.

Para ello lo primero que hacen es informarse sobre los distintos sistemas de almacenamiento posibles y sus ventajas a la hora de utilizarlos con la tecnología **XML**.

En este proceso **Félix descubre que, además de las bases de datos relacionales, existen las orientadas a objetos y las nativas.**

En lo que concierne a las bases de datos, **XML** permite integrar sistemas de información hasta ahora separados:

- **Sistemas de información basados en documentos** (ficheros), tienen estructura irregular, utilizan tipos de datos relativamente simples y dan gran importancia al orden.
- **Sistemas de información estructurados** (bases de datos relacionales), son relativamente planos, utilizan tipos de datos relativamente complejos y dan poca importancia al orden.

Jorge glez. - www.flickr.com (CC BY-NC)

Podemos establecer las siguientes semejanzas entre una base de datos y un fichero **XML** con su esquema asociado:

- La tecnología **XML** usa uno o más documentos para almacenar la información.
- Define esquemas sobre la información.
- Tiene lenguajes de consulta específicos para recuperar la información requerida.
- Dispone de **APIs** (**SAX**, **DOM**).

Pero aparecen muchas más cosas que lo diferencian. Debido a que no es una base de datos, la tecnología **XML** carece, entre otras cosas, tanto de almacenamiento y actualización eficientes como de índices, seguridad, transacciones, integridad de datos, acceso concurrente, disparadores, etc.; que son algunas de las características habituales en las bases de datos. Por tanto es imposible pensar que **XML** se vaya a utilizar para las tareas transaccionales de una organización para las cuales sigue estando sobradamente más justificado utilizar una base de datos.

Un ejemplo de esto es **JABX (Java Architecture for XML Binding)** del que puedes obtener más información en el siguiente enlace.

Autoevaluación

Marcar las afirmaciones que son correctas sobre bases de datos XML:

- [ ] Integran sistemas de información basados en documentos.
- [ ] Integran sistemas de información estructurados.
- [ ] La información se estructura siguiendo esquemas definidos previamente.
- [ ] Garantiza la actualización eficiente.

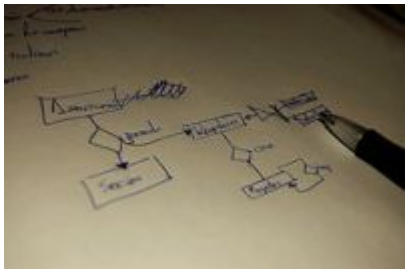
Mostrar retroalimentación

Solución

1. Correcto
2. Correcto
3. Correcto
4. Incorrecto

3.- XML y BD Relacionales.

Las Bases de Datos Relacionales se basan en las relaciones (tablas bidimensionales), como único medio para representar los datos del mundo real. Están asociadas al lenguaje estándar **SQL**.



-ELFisgon- www.flickr.com (CC BY-NC-ND)

Se han creado complejas teorías y patrones para encajar objetos o estructuras jerarquizadas en bases de datos relacionales. Existen numerosos middlewares encargados de la transferencia de información entre estructuras **XML** y bases de datos relacionales.

Las Bases de Datos Relacionales suponen una posibilidad para el almacenamiento de datos **XML**. Sin embargo, no están bien preparadas para almacenar estructuras de tipo jerárquico como son los documentos **XML**, algunas de las causas son:

- Las bases de datos relacionales tienen una estructura regular frente al carácter heterogéneo de los documentos **XML**.
- Los documentos **XML** suelen contener muchos niveles de anidamiento mientras que los datos relacionales son planos.
- Los documentos **XML** tienen un orden intrínseco mientras que los datos relacionales son no ordenados.
- Los datos relacionales son generalmente densos (cada columna tiene un valor), mientras que los datos **XML** son dispersos, es decir, pueden representar la carencia de información mediante la ausencia del elemento.

Algunas de las razones para usar los tipos de Bases de Datos Relacionales y los productos de bases de datos existentes para almacenar **XML**, aún cuando no sea de forma nativa son:

- Las bases de datos relacionales y orientadas a objetos son bien conocidas, mientras que las bases de datos **XML** nativas son nuevas.
- Como resultado de la familiaridad con las bases de datos relacionales y orientadas a objetos, los usuarios se inclinan a ellas especialmente por el rendimiento.

Autoevaluación

El hecho de que el uso de las bases de datos relacionales esté muy extendido es uno de los motivos por los que se utilizan para almacenar datos XML.

- ○ Sí.
- ○ No.

Efectivamente es correcto, al ser familiares para el usuario éste prefiere optar por su uso.

Dado que son muy conocidas son las más fáciles de implementar.

Solución

1. Opción correcta
2. Incorrecto

3.1.- De DB Relacional a XML.

El proceso de traducción puede ser descompuesto en los siguientes pasos básicos:

Leo Reynolds - www.flickr.com (CC BY-NC-SA)

- **Crear el esquema XML** con un elemento para cada tabla y los atributos correspondientes para cada columna no clave. Las columnas que no permiten valores nulos pueden ser marcadas como requeridas, mientras que aquellas que permiten valores nulos pueden ser marcadas como opcionales en el esquema **XML**. Las columnas pueden ser también anidadas como elementos, pero pueden surgir problemas cuando el mismo nombre de columna es usado en más de una tabla. Por ello, lo más simple es transformar las columnas como atributos **XML**, donde las colisiones de nombre en el esquema **XML** no son un problema.

- **Crear las claves primarias en el esquema XML**. Una solución podría ser agregar un atributo para la columna clave primaria, con un ID agregado al nombre de la columna. Este atributo podría necesitar ser definido en el esquema **XML** como de tipo ID. Pueden surgir problemas de colisión al crear claves primarias en el esquema **XML**, ya que a diferencia de las bases de datos relacionales, donde las claves primarias necesitan ser únicas sólo dentro de una tabla, un atributo ID dentro de un documento **XML** debe ser único a través de todo el documento. Para resolverlo se puede agregar el nombre del elemento (nombre de la tabla), al valor de la clave primaria (valor del atributo). Esto asegura que el valor es único a través del documento **XML**.

- **Establecer las relaciones de clave migrada**. Esto se puede lograr mediante el anidamiento de elementos bajo el elemento padre, un ID de esquema **XML** puede ser usado para apuntar a una estructura **XML** correspondiente conteniendo un **`IDREF`**.

Pueden existir muchas variaciones de esquemas **XML** para representar la misma base de datos relacional.

Autoevaluación

Ordena los siguientes pasos con su orden a la hora de transformar una base de datos relacional a almacenamiento XML:

a. **Establecer las relaciones de clave migrada.**
b. **Crear el esquema XML**
c. **Crear las claves primarias**

Indica en cada caso la letra de la opción elegida.

Primer lugar: ☐
Segundo lugar: ☐
Tercer lugar: ☐

Enviar

El orden adecuado es crear el esquema, luego las claves primarias y por último las migradas.

4.- XML y BD Orientadas a Objetos.

reiven - www.flickr.com (CC BY-NC-SA)

Las **bases de datos orientadas a objetos** (**DBOO**) soportan un modelo de objetos puro, en el sentido de que no están basados en extensiones de otros modelos más clásicos como el relacional:

- Están influenciados por los lenguajes de programación orientados a objetos.
- Pueden verse como un intento de añadir la funcionalidad de un **SGBD** a un lenguaje de programación.
- Son una alternativa para el almacenamiento y gestión de documentos **XML**.

Componentes del estándar de Orientación a Objetos:

- **Modelo de Objetos.** Está concebido para proporcionar un modelo de objetos estándar para las bases de datos orientadas a objetos. Es el modelo en el que se basan el los demás componentes.
- **Lenguajes de Especificación de Objetos (ODL).** Para definir los objetos.
- **Lenguaje de Consulta de Objetos (OQL).** Para realizar consultas contra los objetos.
- **‘ Bindings’ para C++, Java y Smalltalk.** Definen un Lenguaje de Manipulación de Objetos (**OML**) que extiende el lenguaje de programación para soportar objetos persistentes. Además incluyen soporte para **OQL**, navegación y transacciones.

Una vez transformado el documento **XML** en objetos, éstos son gestionados directamente por el **SGBDOO**. Dicha información se consulta acudiendo al lenguaje de consulta **OQL**. Los mecanismos de indexación, optimización, procesamiento de consultas, etc. son los del propio **SGBDOO**, y por lo general, no son específicos para el modelo **XML**.

Autoevaluación

Selecciona aquellos que sean componentes del estándar de objetos:

- ○ SQL.
- ○ OQL.
- ○ XQuery.
- ○ XQL.

No es correcta porque es el lenguaje declarativo asociado a las bases de datos relacionales.

Muy bien. Esto lo has entendido, pasa al siguiente apartado.

Es incorrecta porque es el lenguaje declarativo asociado a las bases de datos nativas.

No es correcta pues no existe ningún lenguaje con ese nombre.

Solución

1. Incorrecto
2. Opción correcta
3. Incorrecto
4. Incorrecto

5.- BD XML Nativas.

Las Bases de Datos **XML** Nativas son bases de datos (y como tales soportan transacciones, acceso multi-usuario, lenguajes de consulta, etc) diseñadas especialmente para almacenar documentos **XML**.

adesigna - www.flickr.com (CC BY-NC-SA)

Las **BD XML** Nativas se caracterizan principalmente por:

- **Almacenamiento de documentos en colecciones.** Las colecciones juegan en las bases de datos nativas el papel de las tablas en las **BD** relacionales.
- **Validación de los documentos.**
- **Consultas.** La mayoría de las **BD XML** Nativas soportan uno o más lenguajes de consulta. Uno de los más populares es **XQuery**.
- **Indexación XML.** Se ha de permitir la creación de índices que aceleren las consultas realizadas.
- **Creación de identificadores únicos.** A cada documento **XML** se le asocia un identificador único.
- **Actualizaciones y Borrados**.

Según el tipo de almacenamiento utilizado pueden dividirse en dos grupos:

- **Almacenamiento Basado en Texto.** Almacena el documento **XML** entero en forma de texto y proporciona alguna funcionalidad de base de datos para acceder a él. Hay dos posibilidades:
  - **Posibilidad 1**: Almacenar el documento como un **BL0B** en una base de datos relacional, mediante un fichero, y proporcionar algunos índices sobre el documento que aceleren el acceso a la información.
  - **Posibilidad 2**: Almacenar el documento en un almacén adecuado con índices, soporte para transacciones, etc.
- **Almacenamiento Basado en el Modelo.** Almacena un modelo binario del documento (por ejemplo, DOM) en un almacén existente o bien específico.
  - **Posibilidad 1**: Traducir el DOM a tablas relacionales como Elementos, Atributos, Entidades, etc.
  - **Posibilidad 2**: Traducir el DOM a objetos en una **BDOO**.
  - **Posibilidad 3**: Utilizar un almacén creado especialmente para esta finalidad.

Autoevaluación

Selecciona aquellas características de las bases de datos nativas:

- [ ] Triggers.

- [ ] Consultas.

- [ ] Creación de índices.

- [ ] Creación de identificadores únicos.

Mostrar retroalimentación

Solución

1. Incorrecto
2. Correcto
3. Correcto
4. Correcto

6.- XQuery.

Caso práctico

Una vez que María vuelve de vacaciones Juan y **Félix** le ponen al día de sus estudios.

La única cuestión que **queda por decidir es el modo de acceder a los datos guardados en los archivos XML.**Juan les explica que existe un lenguaje semejante al **SQL**, con el que están familiarizados, llamado **XQuery** y que a su vez está basado en **XPath**.

XQuery es un lenguaje diseñado para escribir consultas sobre colecciones de datos expresadas en **XML**. Puede aplicarse tanto a archivos **XML**, como a bases de datos relacionales con funciones de conversión de registros a **XML**. Su principal función es extraer información de un conjunto de datos organizados como un árbol de etiquetas **XML**. En este sentido **XQuery** es independiente del origen de los datos.

Permite la construcción de expresiones complejas combinando expresiones simples de una manera muy flexible.

Anne Helmond - www.flickr.com (CC BY-NC-ND)

De manera general podemos decir que **XQuery es a XML lo mismo que SQL es a las bases de datos relacionales**. Al igual que éste último, XQuery es un lenguaje funcional.

Los r**equerimientos técnicos** más importantes **de XQuery** se detallan a continuación:

- Debe ser un **lenguaje declarativo**.
- Debe ser i**ndependiente del protocolo de acceso** a la colección de datos. Esto significa que una consulta en **XQuery**, debe funcionar igual al consultar un archivo local, que al consultar un servidor de bases de datos, o que al consultar un archivo **XML** en un servidor web.
- Las consultas y los resultados deben respetar el **modelo de datos XML**.
- Las consultas y los resultados deben ofrecer **soporte para los namespaces.**
- Debe soportar **XML-Schemas y DTDs** y también debe ser capaz de trabajar sin ellos.
- Ha de ser **independiente de la estructura** del documento, esto es, funcionar sin conocerla.
- Debe soportar **tipos simples**, como enteros y cadenas, y **tipos complejos**, como un nodo compuesto.
- Las consultas deben soportar **cuantificadores universales** (para todo) y existenciales (existe).

- Las consultas deben soportar **operaciones sobre jerarquías de nodos** y secuencias de nodos.
- Debe ser posible **combinar información de múltiples fuentes** en una consulta.
- Las consultas deben ser capaces de **manipular los datos independientemente del origen** de estos.
- **El lenguaje de consulta debe ser independiente de la sintaxis**, esto es, pueden existir varias sintaxis distintas para expresar una misma consulta en **XQuery**.

6.1.- Aplicaciones.

Gory - www.flickr.com (CC BY-NC-SA)

Una vez que hemos visto la definición del lenguaje y sus principales requerimientos, queda pensar, ¿para qué se utiliza?

Sus principales aplicaciones se resumen en tres:

- **Recuperar información a partir de conjuntos de datos XML**.
- **Transformar unas estructuras de datos XML** en otras estructuras que organizan la información de forma diferente.
- **Ofrecer una alternativa a XSLT** para realizar transformaciones de datos en **XML** a otro tipo de representaciones, como **HTML** o **PDF**.

¿Y cuáles son los motores **XQuery** de código abierto más relevantes y sus características principales?

- **BaseX**: proyecto open-source, con interfaz gráfica y disponible para *Linux*, *Windows* y *Mac*.
- **Qexo:** escrito en Java y con licencia GPL que se distribuye integrado dentro del paquete Kawa.
- **Saxon:** escrito en Java y distribuido en múltiples paquetes, algunos open-source y otros bajo licencia comercial.

Autoevaluación

¿Cuáles de las siguientes herramientas se relacionan directamente con XQuery?

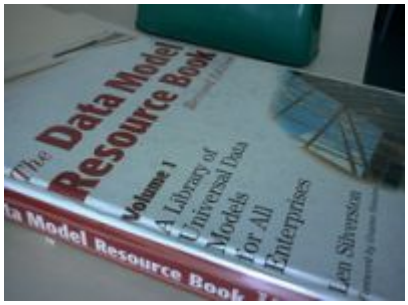
- [ ] Xquark-Brigde.
- [ ] Oracle.
- [ ] Saxon.
- [ ] MySQL.

Mostrar retroalimentación

Solución

1. Correcto
2. Incorrecto
3. Correcto
4. Incorrecto

6.2.- Modelo de datos.



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** el orden en que se encuentren los datos es importante, ya que no es lo mismo buscar una etiqueta **<B>** dentro de una etiqueta **<A>** que todas las etiquetas **<B>** del documento (que pueden estar anidadas dentro de una etiqueta **<A>** o no).

La entrada y la salida de una consulta **XQuery** se define en términos de un modelo de datos. Dicho modelo de datos de la consulta proporciona una representación abstracta de uno o más documentos **XML** (o fragmentos de documentos).

Las principales características de este modelo de datos son:

- Se basa en la **definición de secuencia**, como una colección ordenada de cero o más **`ítems`**. Éstas pueden ser heterogéneas, es decir pueden contener varios tipos de nodos y valores atómicos. Sin embargo, una secuencia nunca puede ser un ítem de otra secuencia.
- **Orden del documento**: corresponde al orden en que los nodos aparecerían si la jerarquía de nodos fuese representada en formato **XML**, (si el primer carácter de un nodo ocurre antes que el primer carácter de otro nodo, lo precederá también en el orden del documento).
- Contempla un **valor especial llamado "error value"** que es el resultado de evaluar una expresión que contiene un error.

Autoevaluación

Alguna de las características del modelo de datos en el que se basa XQuery es:

- ○ El orden de los nodos no importa.
- ○ Se basa en secuencias de ítems, donde las secuencias se anidan dentro de otras secuencias.
- ○ La secuencia es un conjunto de datos de cualquier tipo.
- ○ Tiene un valor especial que aparece cuando se trata de evaluar una expresión errónea.

No es correcta porque mantiene la jerarquía de los datos representados en XML.

Incorrecta, porque una secuencia nunca puede ser un ítem de otra secuencia.

No es la respuesta correcta, porque el conjunto ha de estar ordenado.

Efectivamente es correcto, "error value" aparece cuando se intenta evaluar una consulta con errores.

Solución

1. Incorrecto
2. Incorrecto
3. Incorrecto
4. Opción correcta

Debes conocer

En el siguiente enlace puedes encontrar el estándar de XQuery aprobado por el W3C. La última recomendación es de marzo de 2017.

Recomendación del W3C sobre XQuery version 3.1.

6.3.- Expresiones.

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

seferman - www.flickr.com (CC BY-NC-SA)

El valor de una expresión es una secuencia heterogénea de nodos y valores atómicos.
La mayoría de las expresiones están compuestas por la combinación de expresiones más simples unidas mediante operadores y palabras reservadas.

Ya hemos visto que **XPath** es un lenguaje declarativo para la localización de nodos y fragmentos de información en árboles **XML**.
Puesto que **XQuery** ha sido construido sobre la base de **XPath** y realiza la selección de información y la iteración a través del conjunto de datos basándose en dicho lenguaje, **toda expresión XPath también es una consulta Xquery válida**.

Los **comentarios** en **XQuery** están limitados entre caras sonrientes, es decir: **`(: Esto es un comentario XQuery :)`**.

En un documento **XQuery los caracteres { } delimitan las expresiones que son evaluadas** para crear un documento nuevo.

XQuery admite expresiones condicionales del tipo **`if-then-else`** con la misma semántica que tienen en los lenguajes de programación habituales.

Las consultas **XQuery** pueden estar formadas por hasta cinco tipos de cláusulas diferentes, siguen la norma **FLWOR** (que se pronuncia "flower"). Estas cláusulas son los bloques principales del **XQuery**, equivalen a las cláusulas **select, from, where, group by, having, order by** y **limit** de **SQL.**

En una sentencia **FLWOR al menos ha de existir una cláusula FOR o una LET**, el resto, si existen, han de respetar escrupulosamente el orden dado por el nombre, **FLWOR**.
Con estas sentencias se consigue buena parte de la funcionalidad que diferencia a **XQuery** de **XPath**. Entre otras cosas permite construir el documento que será la salida de la sentencia.

Una consulta **XQuery** está formada por dos partes:

- **Prólogo**: Lugar donde se declaran los espacios de nombres, de funciones, variables, etc.
- **Expresión**: Consulta propiamente dicha.

Autoevaluación

Toda sentencia XPath es una sentencia XQuery:

- ○ Verdadero.

○ Falso.

Muy bien, has captado la idea.

Incorrecta, ya que XQuery se basa en XPath.

Solución

1. Opción correcta
2. Incorrecto

6.4.- Claúsulas.

Hemos visto el modo de crear sentencias **FLOWR**, vamos ahora a estudiar aisladamente cada una de las cláusulas que pueden formar estas sentencias.

Galería de JoshuaDavisPhotography - www.flickr.com (CC BY-SA)

- **FOR**: asocia una o más variables con cada nodo que encuentre en la colección de datos. Si en la consulta aparece más de una cláusula **FOR** (o más de una variable en una cláusula **FOR**), el resultado es el producto cartesiano de dichas variables.
- **LET**: vincula las variables al resultado de una expresión. Si esta cláusula aparece en una sentencia en la que ya hay al menos una cláusula **FOR**, los valores de la variable vinculada por la cláusula **LET** se añaden a cada una de las tuplas generadas por la cláusula **FOR**.
- **WHERE**: filtra tuplas producidas por las cláusulas **FOR** y **LET**, quedando solo aquellas que cumplen con la condición.
- **ORDER BY**: ordena las tuplas generadas por **FOR** y **LET** después de que han sido filtradas por la cláusula **WHERE**. Por defecto el orden es ascendiente, pero se puede usar el modificador **descending** para cambiar el sentido del orden.
- **RETURN:** construye el resultado de la expresión **FLWOR** para una tupla dada.

Debes conocer

En el siguiente enlace puedes ver una presentación que muestra el modo en el que se ejecuta una sentencia FLWOR.

Ejecución de una consulta XQuery.

https://www.youtube.com/embed/0fR4Twl9las

Resumen textual alternativo

6.5.- Ejemplos XQuery.

Vamos a ver algunos ejemplos de XQuery utilizando el fichero libros.xml. En este fichero:

- El elemento raíz es **biblioteca**. Contiene un elemento libros.
- Dentro de libros, hay varios elementos libro.
- Los elementos **libro** tienen atributos **publicacion** y **edicion** (opcional). También tienen elementos titulo, autor (puede haber más de uno), **editorial**, **paginas** y un elemento opcional para indicar si hay edición electrónica, **edicionElectronica**.

Ejercicio Resuelto 1

1.- Título y editorial de todos los librosPara devolver varios campos, los envolvemos en un elemento.

Mostrar retroalimentación

```
for $x in doc("libros.xml")/biblioteca/libros/libro
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
  <titulo>Learning XML</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XML Imprescindible</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XML Schema</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XPath Essentials</titulo>
  <editorial>Wiley</editorial>
</libro>
<libro>
  <titulo>Beginning XSLT 2.0: Form Novice to Professional</titulo>
  <editorial>Apress</editorial>
</libro>
<libro>
  <titulo>XQuery</titulo>
  <editorial>O'Reilly</editorial>
</libro>
```

Ejercicio Resuelto 2

2.- El título (sin etiquetas) de todos los libros de menos de 100 páginas.Para hacer comparaciones con números, lo mejor es convertir los datos con la función **`number`** para evitar problemas de tipo de dato o que los compare como cadenas.

Mostrar retroalimentación

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where number($x/paginas) < 100
return data($x/titulo)
```

```
Learning XML
```

Ejercicio Resuelto 3

3.- El número de libros de menos de 100 páginas.Utilizamos la función **`count().`**

Mostrar retroalimentación

```
for $x in doc("libros.xml")/biblioteca/libros
let $y := $x/libro[number(paginas) < 100]
return count($y)
```

```
1
```

Ejercicio Resuelto 4

4.- Una lista HTML con el título de los libros de la editorial “O'Reilly” ordenados por título.Podemos mezclar etiquetas HTML y XQuery y obtener HTML como resultado de una consulta.

Mostrar retroalimentación

```
<ul>
{
for $x in doc("libros.xml")/biblioteca/libros/libro
where $x/editorial = "O'Reilly"
order by $x/titulo
return <li>{data($x/titulo)}</li>
}
</ul>
```

```
<ul>
  <li>Learning XML</li>
  <li>XML Imprescindible</li>
  <li>XML Schema</li>
  <li>XQuery</li>
</ul>
```

Ejercicio Resuelto 5

5.- Título y editorial de los libros de 2002.

Mostrar retroalimentación

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where $x[@publicacion=2002]
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
  <titulo>XML Schema</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XPath Essentials</titulo>
  <editorial>Wiley</editorial>
</libro>
```

Ejercicio Resuelto 6

6.- Título y editorial de los libros con más de un autor.

Mostrar retroalimentación

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where count($x/autor)>1
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
  <titulo>XML Imprescindible</titulo>
  <editorial>O'Reilly</editorial>
</libro>
```

Ejercicio Resuelto 7

7.- Título y editorial de los libros que tienen versión electrónica.

Mostrar retroalimentación

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where $x/versionElectronica
return <libro>{$x/titulo, $x/editorial}</libro>
```

```
<libro>
  <titulo>Learning XML</titulo>
  <editorial>O'Reilly</editorial>
</libro>
<libro>
  <titulo>XPath Essentials</titulo>
  <editorial>Wiley</editorial>
</libro>
```

Ejercicio Resuelto 8

8.- Título de los libros que no tienen versión electrónica.

Mostrar retroalimentación

```
for $x in doc("libros.xml")/biblioteca/libros/libro
where not($x/versionElectronica)
return$x/titulo
```

```
<titulo>XML Imprescindible</titulo>
<titulo>XML Schema</titulo>
<titulo>Beginning XSLT 2.0: Form Novice to Professional</titulo>
<titulo>XQuery</titulo>
```

6.6.- Funciones.

Ahora que conocemos las cláusulas que sustentan el lenguaje **XQuery**, vamos a ocuparnos de las **funciones que soporta.**
Estas son funciones matemáticas, de cadenas, para el tratamiento de expresiones regulares, comparaciones de fechas y horas, manipulación de nodos XML, manipulación de secuencias, comprobación y conversión de tipos y lógica booleana. Además permite definir funciones propias y funciones dependientes del entorno de ejecución del motor **XQuery**. Las funciones más importantes se muestran a continuación:

- **Funciones numéricas**
  - **`floor()`**, que devuelve el valor numérico inferior más próximo al dado.
  - **`ceiling()`**, que devuelve el valor numérico superior más próximo al dado.
  - **`round()`**, que redondea el valor dado al más próximo.
  - **`count()`**, determina el número de **`ítems`** en una colección.
  - **`min()`** o **`max()`**, devuelven respectivamente el mínimo y el máximo de los valores de los nodos dados.
  - **`avg()`**, calcula el valor medio de los valores dados.
  - **`sum()`**, calcula la suma total de una cantidad de **`ítems`** dados.

- **Funciones de cadenas de texto**
  - **`concat()`**, devuelve una cadena construida por la unión de dos cadenas dadas.
  - **`string-length()`**, devuelve la cantidad de caracteres que forman una cadena.
  - **`startswith()`, `ends-with()`**, determinan si una cadena dada comienza o termina, respectivamente, con otra cadena dada.
  - **`upper-case()`, `lower-case()`**, devuelve la cadena dada en mayúsculas o minúsculas respectivamente.

- **Funciones de uso general**
  - **`empty()`**, devuelve "**`true`**" cuando la secuencia dada no contiene ningún elemento.
  - **`exits()`**, devuelve "**`true`**" cuando una secuencia contiene, al menos, un elemento.
  - **`distinct-values()`**, extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.
  - **`data()`**, devuelve el valor de los elementos que recibe como argumentos, es decir, sin etiquetas.

- **Cuantificadores existenciales:**
  - **`some`**, **`every`**, permiten definir consultas que devuelven algún, o todos los elementos, que verifiquen la condición dada.

Ejemplo de la función data

Esta consulta devuelve todos los títulos, incluyendo las etiquetas:

```
for $x in doc("libros.xml")/biblioteca/libros/libro/titulo
return $x
```

El resultado con el fichero de ejemplo sería:

```
<titulo>Learning XML</titulo>
<titulo>XML Imprescindible</titulo>
<titulo>XML Schema</titulo>
<titulo>XPath Essentials</titulo>
<titulo>Beginning XSLT 2.0: Form Novice to Professional</titulo>
<titulo>XQuery</titulo>
```

Utilizando la función data:

```
for $x in doc("libros.xml")/biblioteca/libros/libro/titulo
return data($x)
```

...se obtiene:

```
Learning XML
XML Imprescindible
XML Schema
XPath Essentials
Beginning XSLT 2.0: Form Novice to Professional
XQuery
```

Además de estas funciones que están definidas en el lenguaje, **XQuery** permite al desarrollador construir sus propias funciones:

```
declare nombre_funcion($param1 as tipo_dato1, $param2 as tipo_dato2,… $paramN as tipo_dat
as tipo_dato_devuelto
{
...CÓDIGO DE LA FUNCIÓN...
}
```

6.7.- Ejemplo: definición y llamada a una función.

Ejercicio Resuelto

En este ejemplo puedes ver la definición y llamada a una función escrita por el usuario que nos calcula el precio de un libro una vez que se le ha aplicado el descuento.

Mostrar retroalimentación

```
declare function minPrice($p as xs:decimal?,$d as xs:decimal?) as xs:
  {
  let $disc := ($p * $d) div 100
  return ($p - $disc)
  }
```

Un ejemplo de cómo invocar a la función desde la consulta es:

```
<minPrice>{minPrice($libros/precio,$libros/descuento)}</minPrice>
```

6.8.- Operadores.

Veamos ahora algunos de los operadores más importantes agrupados según su funcionalidad:

- **Comparación de valores:** Comparan dos valores escalares y produce un error si alguno de los operandos es una secuencia de longitud mayor de 1. Estos operadores son:
  - **`eq`**, igual.
  - **`ne`**, no igual.
  - **`lt`**, menor que.
  - **`le`**, menor o igual que.
  - **`gt`**, mayor que.
  - **`ge`**, mayor o igual que.
- **Comparación generales:** Permiten comparar operandos que sean secuencias.
  - **=**, igual.
  - **!=**, distinto.
  - **>**, mayor que.
  - **>=**, mayor o igual que.
  - **<,** menor que.
  - **<=**, menor o igual que.
- **Comparación de nodos:** Comparan la identidad de dos nodos.
  - **`is`**, devuelve true si las dos variables que actúan de operandos están ligadas al mismo nodo.
  - **`is not`**, devuelve true si las dos variables no están ligadas al mismo nodo.
- **Comparación de órdenes de los nodos: <<**, compara la posición de dos nodos. Devuelve "**true**" si el nodo ligado al primer operando ocurre primero en el orden del documento que el nodo ligado al segundo.
- **Lógicos: `and`** y **`or`** Se emplean para combinar condiciones lógicas dentro de un predicado.
- **Secuencias de nodos:** Devuelven secuencias de nodos en el orden del documento y eliminan duplicados de las secuencias resultado.
  - **Unión**, devuelve una secuencia que contiene todos los nodos que aparecen en alguno de los dos operandos que recibe.
  - **`Intersect`**, devuelve una secuencia que contiene todos los nodos que aparecen en los dos operandos que recibe.
  - **`Except`**, devuelve una secuencia que contiene todos los nodos que aparecen en el primer operando que recibe y que no aparecen en el segundo.
- **Aritméticos**: **+, -, *, `div` y `mod`**, devuelven respectivamente la suma, diferencia, producto, cociente y resto de operar dos números dados.

Autoevaluación

Cuáles de los siguientes operadores pertenecen a SQL y a XQuery:

- [ ] is.

- [ ] union.

- [ ] lt.

- [ ] >=.

Mostrar retroalimentación

Solución

1. Correcto
2. Correcto
3. Incorrecto
4. Correcto

7.- Para saber más.

Para saber más

Información adicional:

- Tutorial XQuery -W3School.
- XLS, XPath y XQuery referencia de funciones.