XML y Bases de Datos

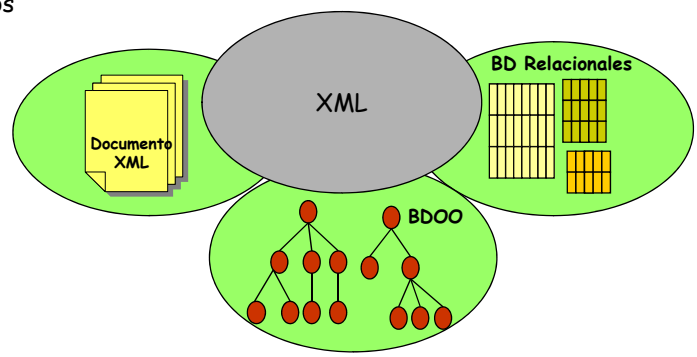
Ana Belén Martínez Prieto
Universidad de Oviedo

Contenidos

- Introducción
- XML y Bases de Datos
- XQuery

XML – Introducción

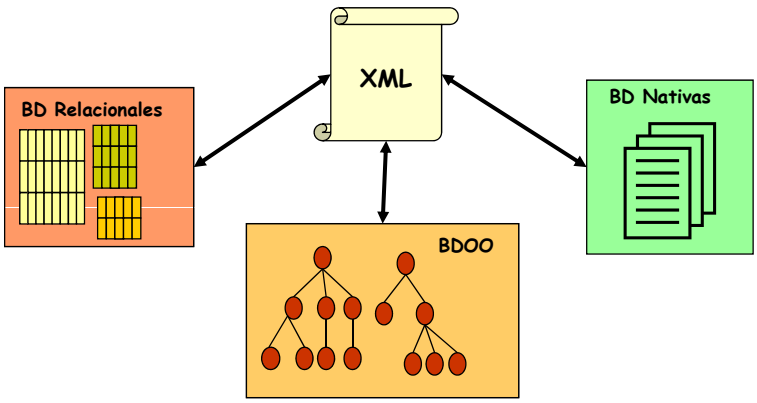
- Estándar para el intercambio de datos entre aplicaciones en Internet independiente del formato de almacenamiento de los mismos



- Es lógico que las consultas entre aplicaciones se expresen como consultas contra los datos en formato XML → XQuery

XML – Introducción (II)

- Muchas aplicaciones requieren el almacenamiento de datos XML
- Existen diferentes alternativas:



Contenidos

- Introducción
- XML y Bases de Datos
  - **Bases de Datos Relacionales**
  - **Bases de Datos Orientadas a Objetos**
  - **Bases de Datos Nativas**
- XQuery

Bases de Datos Relacionales

- Se basan en las relaciones (tablas bidimensionales) como único medio para representar los datos del mundo real
- Lenguaje estándar **SQL**

Atributos->Dominio

Relación Alumnos

DNI	Nombre	Dirección	FNac
10345678	Juan González Suárez	Uría 27, 2ºA	19-06-1980
76987654	Sonia García Martínez	Begoña 34, 1ºB	16-05-1983
09875432	María Pérez Pérez	Av. Galicia 12, 6ºB	18-04-1983

Relación Asignaturas

Código	Nombre	Créditos	Curso	Tipo
01	Bases de Datos	9	3	Obligatoria
02	Programación	9	1	Troncal
03	Álgebra	6	1	Obligatoria

Relación AlumAsig

DNI	CodigoAsig	Nota
10345678	01	4
10345678	02	3
09875432	01	7

Tupla

Bases de Datos Relacionales (II)

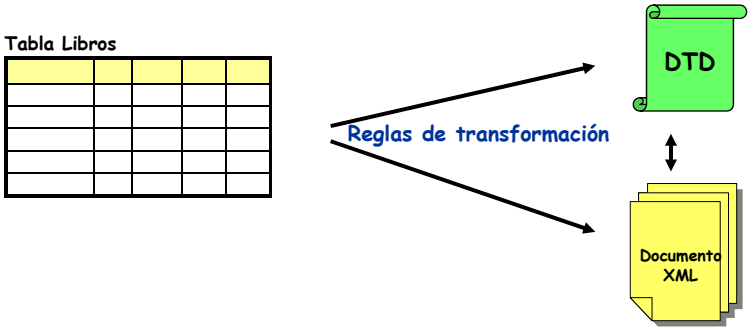
- Se han creado complejas teorías y patrones para encajar objetos o estructuras jerarquizadas en bases de datos relacionales
  - Existen **numerosos middlewares** encargados de la transferencia de información entre estructuras XML y bases de datos relacionales

Nombre	Fabricante	Licencia	XML->BD	BD->XML
ADO	Microsoft	Comercial	X	X
Allora	HiT Software	Comercial	X	X
ASP2XML	Stonebroom	Comercial	X	X
Data Junction	Data Junction Inc.	Open Source	X	X
DBX	Swift Inc.	Comercial	X	X
InterAccess	XML Soft. Corporation	Comercial	X	X
JaxMe	Jochen Wiedmann	Open Source	X	X
XML-DBMS	Ronald Bourret	Open Source	X	X

Bases de Datos Relacionales

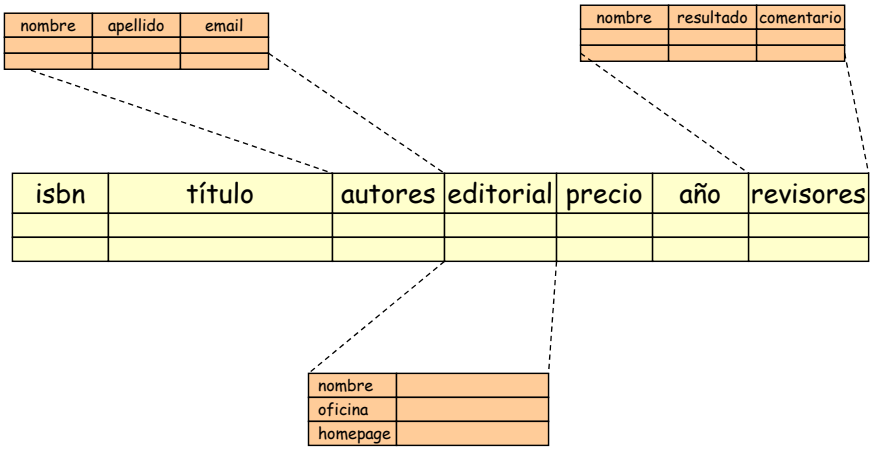
Transformación a XML

- Veamos un ejemplo de transformación de una tabla a un documento XML



Bases de Datos Relacionales

Transformación a XML - Tabla Libros



Bases de Datos Relacionales

Reglas de Transformación Relacional - DTD

- Regla 1
  - **Para cada tabla** en el esquema de la base de datos hay que **crear un elemento** con el mismo **nombre de la tabla** y la cardinalidad apropiada.

```
<!DOCTYPE libros [
        <!ELEMENT libros (libro)*>

]>
```

Bases de Datos Relacionales

Reglas de Transformación Relacional - DTD

- Regla 2
  - Las columnas de la tabla son incluidas en otro elemento (sub-elemento del elemento creado en la regla anterior), que representa un **registro en la tabla**

```
<!ELEMENT libro (isbn, titulo, autores, editor, precio, año, revisores)>
```


Bases de Datos Relacionales

Reglas de Transformación Relacional - DTD

- Regla 3
  - Para cada columna en la tabla cuyo tipo de dato es simple (char, integer, etc.) crear un elemento, sub-elemento del elemento creado en el paso anterior, de tipo #PCDATA con el mismo nombre de la columna.

```
<!ELEMENT isbn (#PCDATA)>
<!ELEMENT titulo (#PCDATA)>
```

Bases de Datos Relacionales

Reglas de Transformación Relacional - DTD

- Regla 4
  - Para cada columna en la tabla cuyo tipo de dato es complejo (tipo objeto), crear un elemento complejo, sub-elemento del elemento creado en el paso 2, con el mismo nombre de la columna. Para cada propiedad del tipo objeto crear un elemento con el mismo nombre de la propiedad.

```
<!ELEMENT autores (autor)+>
        <!ELEMENT autor (nombre, apellidos, email)>
        <!ELEMENT nombre  (#PCDATA)>
        <!ELEMENT apellidos (#PCDATA)>
        <!ELEMENT email (#PCDATA)>
```


Bases de Datos Relacionales

Reglas de Transformación Relacional - DTD

- Regla 5
  - Para cada columna en la tabla que es una tabla anidada, crear un elemento con el mismo nombre de esa columna y la cardinalidad apropiada. Repetir todos los pasos desde el 2.

Bases de Datos Relacionales

DTD Resultante

```
<!DOCTYPE libros [
    <!ELEMENT libros (libro)*>
    <!ELEMENT libro (isbn, titulo, autores, editor, precio, año, revisores)>
    <!ELEMENT isbn (#PCDATA)>
    <!ELEMENT titulo (#PCDATA)>
    <!ELEMENT autores (autor)+>
    <!ELEMENT autor (nombre, apellidos, email)>
    <!ELEMENT editor (nombre,oficina,homepage) >
    <!ELEMENT nombre  (#PCDATA)>
    <!ELEMENT apellidos (#PCDATA)>
    <!ELEMENT email (#PCDATA)>
    <!ELEMENT oficina (#PCDATA)>
    <!ELEMENT homepage (#PCDATA)>
    <!ELEMENT precio (#PCDATA)>
    <!ELEMENT año (#PCDATA)>
    <!ELEMENT revisores (revisor)*>
    <!ELEMENT revisor (nombre,resultado,comentarios)>
    <!ELEMENT nombre (#PCDATA)>
    <!ELEMENT resultado (#PCDATA)>
    <!ELEMENT comentario (#PCDATA)>]>
```


Bases de Datos Relacionales

Posible Documento XML Resultante

```
<?xml version="1.0">
<libros>
  <libro>
      <isbn> 1-55655-767-6 </isbn>
      <titulo> Fundamentos de Bases de Datos </titulo>
      <autores>
          <autor>
              <nombre> Abraham</nombre>
              <apellido> Silberschatz</apellido>
              <email> silbers@hotmail.com </email>
          </autor>
          <autor>
              <nombre> Henry </nombre>
              <apellido> Korth </apellido>
              <email> korth@ hotmail.com </email>
          </autor>
      </autores>
      <editor>
          <nombre> McGraw-Hill </nombre>
          <oficina> Av. Santander s/n </oficina>
          <homepage> http://www.mcgrawhill.es </homepage>
      </editor>
      <precio> 40 </precio>
      <año> 2003 </año>
      <revisores>
          <revisor>
              <nombre> James  Smith </nombre>
              <resultado> 8 </resultado>
              <comentario> Es un libro de texto básico ...</comentario>
          </revisor>
      </revisores>
  </libro>
</libros>
```

Resumen Transformación

Esquema Relacional - DTD

```
<!DOCTYPE tabla [
    <!ELEMENT tabla (registro)*>
    <!ELEMENT registro (columna1, columna2,..., columnaN)>
    <!ELEMENT columna1 (#PCDATA)>
    <!ELEMENT columna2 (propiedad1, propiedad2,..., propiedadN) >
    . . .
    <!ELEMENT columnaN  (#PCDATA)>
    <!ELEMENT propiedad1 (#PCDATA)>
    <!ELEMENT propiedad2 (#PCDATA)>
    . . .
    <!ELEMENT propiedadN (#PCDATA)> ]>
```


Resumen - Transformación

Esquema Relacional - XML SCHEMA

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="tabla">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="registro" minOccurs="0" maxOccurs="unbounded">
              <xsd:complexType>
                    <xsd:sequence >
                        <xsd:element name="columna1"  type="xsd:positiveInteger"/>
                        <xsd:element name="columna2"/>
                            <xsd:complexType>
                              <xsd:sequence >
                                <xsd:element name="propiedad1" type="xsd:string"/>
                                <xsd:element name="propiedad2" type="xsd:boolean"/>
                                 ....
                                <xsd:element name="propiedadn" type="xsd:positiveInteger"/>
                              </xsd:sequence>
                            </xsd:complexType>
                        </xsd:element >
                        <xsd:element name="columnaN"  type="xsd:string"/>
                    </xsd:sequence >
              </xsd:complexType>
            </xsd:element >
          </xsd:sequence >
        </xsd:complexType>
      </xsd:element >
    </xsd:schema>
```

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:
  - BD relacionales tienen una estructura regular frente al carácter heterogéneo de los documentos XML
  - Documentos XML suelen contener muchos niveles de anidamiento mientras que los datos relacionales son "planos"
  - Documentos XML tienen un orden intrínseco mientras que los datos relacionales son no ordenados
  - Datos relacionales son generalmente "densos" (cada columna tiene un valor) mientras que los datos XML son "dispersos" pueden representar la carencia de información mediante la ausencia del elemento

Contenidos

- Introducción
- XML y Bases de Datos
  - **Bases de Datos Relacionales**
  - **Bases de Datos Orientadas a Objetos**
  - **Bases de Datos Nativas**
- XQuery

Bases de Datos Orientadas a Objetos

Introducción

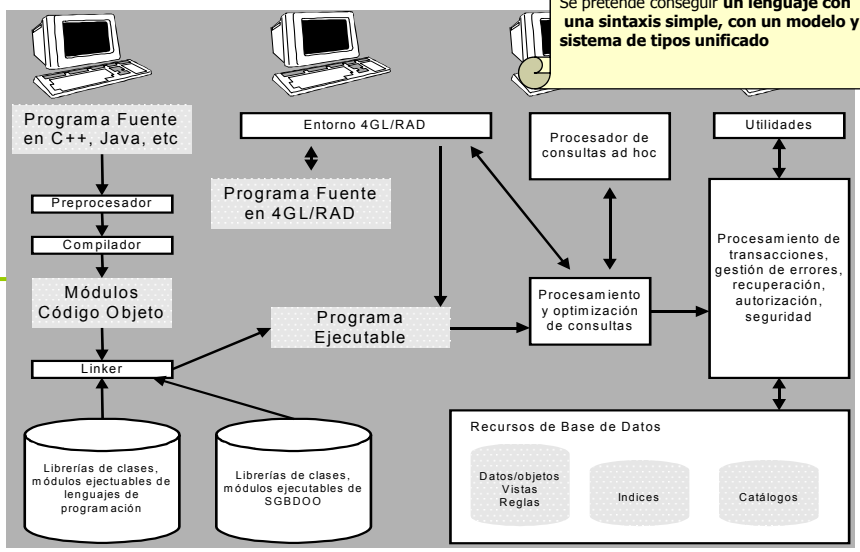
- Los Sistemas de Bases de Datos Orientadas a Objetos soportan un modelo de objetos puro, en la medida de que no están basados en extensiones de otros modelos más clásicos como el relacional:
  - Están fuertemente influenciados por los lenguajes de programación orientados a objetos
  - Pueden verse como un intento de añadir la funcionalidad de un SGBD a un lenguaje de programación
- Ejemplos de SGBDOO son
  - Poet (http://www.poet.com)
  - Jasmine (http://www.cai.com)
  - ObjectStore (http://www.odi.com)
  - GemStone (http://www.gemstone.com)
- Representan **una alternativa viable para el almacenamiento y gestión de documentos XML**

Bases de Datos Orienta

Estructura de Referencia

Extienden el lenguaje de programación y su implementación (compilador, preprocesador, ...) para incorporar funcionalidad de base de datos

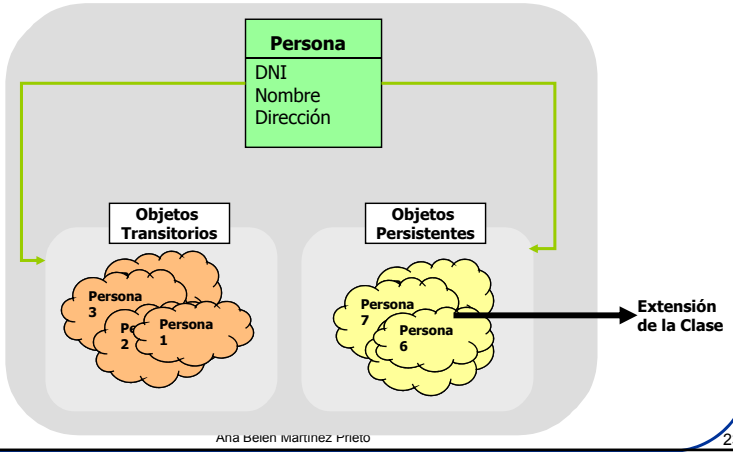
Se pretende conseguir **un lenguaje con una sintaxis simple, con un modelo y sistema de tipos unificado**



Bases de Datos Orientadas a Objetos

Características: Persistencia

- Cualidad de algunos objetos de mantener su identidad y relaciones con otros objetos con independencia del sistema o proceso que los creó.



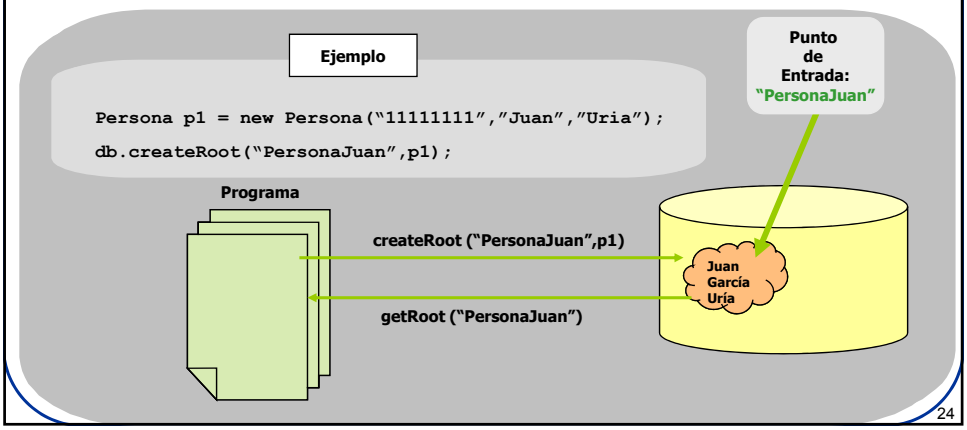
Bases de Datos Orientadas a O

Características: Persistencia por nombramiento

Inconveniente:

El nombrado de cada objeto a nivel individual no resulta práctico para una base de datos de tamaño considerable

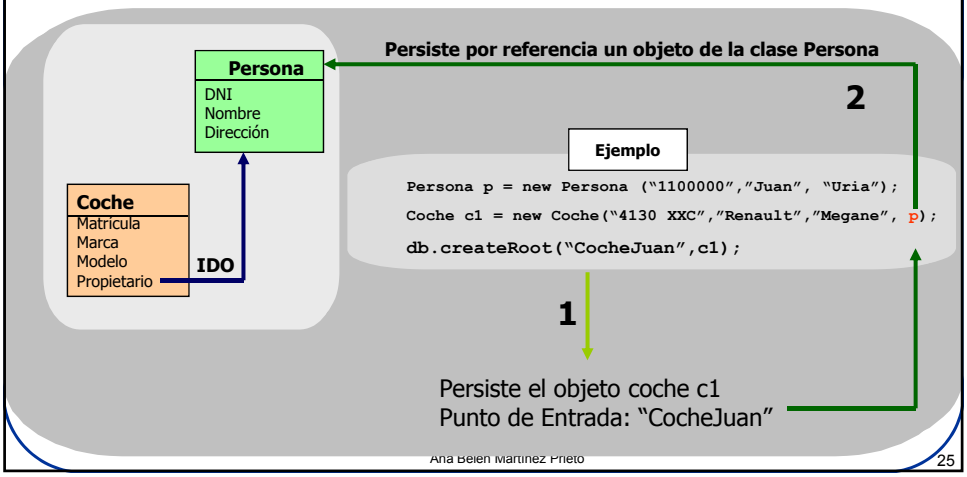
- Por llamada explícita - **nombramiento**
  - Implica dar un nombre único al objeto persistente por el que podrá ser recuperado por otros programas
  - Los nombres han de ser únicos en cada base de datos



Bases de Datos Orientadas a Objetos

Características: Persistencia por alcance

- Por referencia o **alcance**
  - Un objeto se convierte en persistente si es alcanzado desde otro objeto persistente

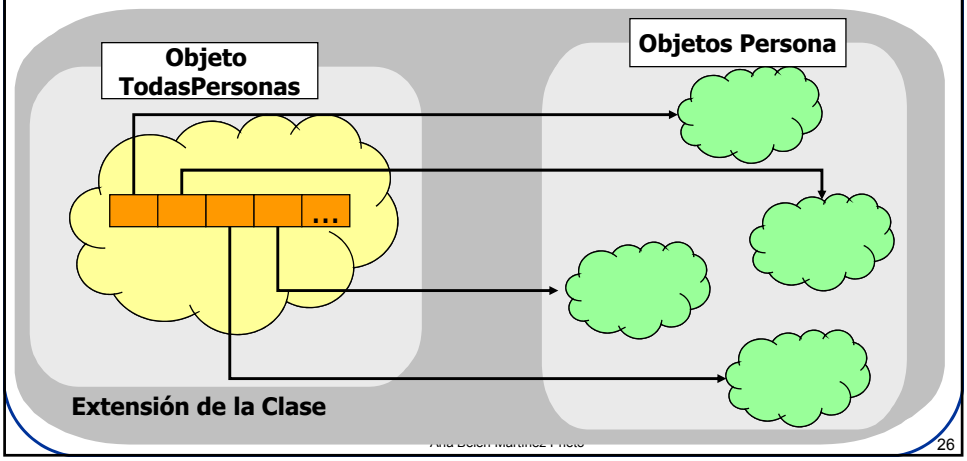



```
TodasPersonas = new OSTreeSet();
db.createRoot("TodasPersonas",TodasPersonas);
.....
Persona p = new Persona("44444444","Pedro","Avilés");
TodasPersonas.add(p);
.....
```

Bases de Datos O

Características: Persistencia por alcance + nombramiento

- Es el mecanismo más utilizado: **Nombramiento y Alcance combinados**
  - Crear un objeto persistente (con nombre) que representa un conjunto o lista de objetos de una determinada clase



Bases de Datos Orientadas a Obje

ODMG 93 - Primera Versión
ODMG 2.0 - Segunda Versión
ODMG 3.0 - Versión actual

http://www.odmg.org

Estándar ODMG

- ODMG (*Object Data Management Group*) representa el estándar para los SGBDOO confeccionado por un conjunto de vendedores de dichos sistemas
- **Componentes del Estándar:**
  - **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 lenguaje de definición de objetos (ODL) y el lenguaje de consultas (OQL)
  - **Lenguajes de Especificación de Objetos**
    - **Lenguaje de Definición de Objetos (ODL**, Object Definition Language)
  - **Lenguaje de Consulta**
    - Conocido como **OQL** (Object Query Language)
    - Lenguaje declarativo estilo SQL
  - **'Bindings' para C++, Java y Smalltalk**
    - Definen un lenguaje de manipulación de objetos (OML, Object Manipulation Language) que extiende el lenguaje de programación para soportar objetos persistentes. Además incluye soporte para OQL, navegación y transacciones

Bases de Datos Orientadas a Objetos

Estándar ODMG-Object Query Language (OQL)

- La sintaxis es similar a la de SQL con características adicionales para conceptos ODMG como:
  - identidad de objetos
  - objetos complejos
  - operaciones
  - herencia
  - polimorfismo y
  - relaciones
- Mantiene la integridad de los datos empleando las operaciones definidas sobre el objeto, en vez de sus propios operadores de actualización

Bases de Datos Orientadas a Objetos
Estándar ODMG-Object Query Language (OQL)- Ejemplos

Expresiones de Camino

```
Select e
From personas p
Where p.direccion.ciudad=Oviedo
```

Invocación de Métodos

```
Select p.nombre
From personas p
Where p.vive_en("Oviedo")
```


Bases de Datos Orientadas a Objetos
Posibilidades ofrecidas para la gestión de XML: Específica

- **Específica**
  - Emplea una nueva clase para cada nuevo tipo de información. Se modelan los datos en el documento XML como un árbol de objetos pero que son específicos para los datos en el documento
  - Esta opción es especialmente apropiada para **documentos XML válidos** y para **aplicaciones centradas en los datos**
  - Existen muchos productos que implementan esta asociación de documentos XML a objetos ***(XML Data Binding) .Ej. JAXB***

Bases de Datos Orientadas a Objetos

Posibilidades ofrecidas para la gestión de XML: Específica

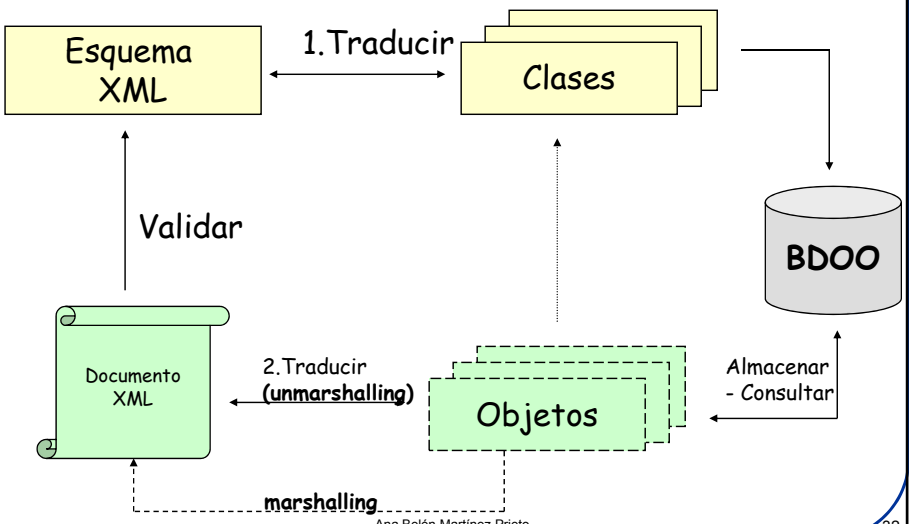
- **XML Data Binding**
  - **JAXB** es el nombre del API oficial de Sun para *bindings* de archivos XML. Sun proporciona una implementación de referencia
  - Herramientas de *binding* para Java:

Nombre Herramienta	URL
Castor	http://www.castor.org
Xgen	http://www.commerceone.com/developers/docsoapxdk
Breeze	http://www.breezefactor.com
Zeus	http://zeus.objectweb.org
XMLBeans	http://xml.apache.org/xmlbeans

- El proceso de *binding* está basado en cuatro conceptos:
  - Generación de código fuente
  - Unmarshalling
  - Marshalling
  - Binding schemas (esquemas de vinculación)

Bases de Datos Orientadas a Objetos

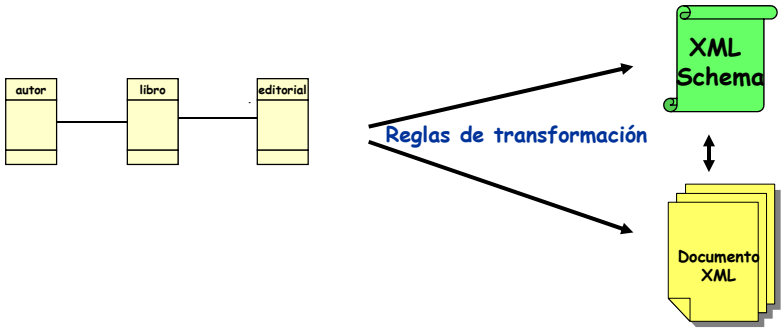
Posibilidades ofrecidas para la gestión de XML: Específica (II)



Bases de Datos Orientadas a Objetos

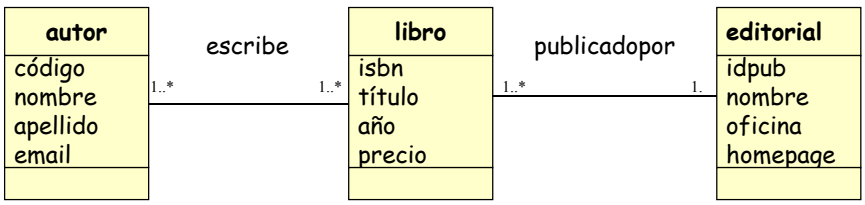
Transformación a XML

- Veamos un ejemplo de transformación de un esquema orientado a objetos a un documento XML



Bases de Datos Orientadas a Objetos

Ejemplo de tranformación de clases a XML Schema



Bases de Datos Orientadas a Objetos
Ejemplo de tranformación – Código ODL

```
Class libro
(extent libros key isbn) {
        attribute string isbn;
        attribute string titulo;
        attribute unsigned short año;
        attribute float precio;
        relationship set <autor> escritopor inverse autor::escribe;
        relationship editorial publicadopor inverse editorial::publica; };

Class autor
(extent autores key codigo) {
        attribute string codigo;
        attribute string nombre;
        attribute string apellido;
        attribute list<string> email;
        relationship set <libro> escribe inverse libro::escritopor; };

Class editorial
(extent editoriales key idpub) {
        attribute string idpub;
        attribute string nombre;
        attribute string oficina;
        attribute string homepage;
        relationship set <libro> publica inverse libro::publicadopor; };
```


Bases de Datos Orientadas a Objetos
Reglas de Transformación ODL – XML Schema

- Regla 1
  - Cada documento XML tiene un **elemento raíz** (puede ser el nombre de la base de datos) que debe tener asociado un **tipo complejo anónimo** que incluye un elemento especial *choice* con un atributo *maxOcurrs* con valor *unbounded*

```
<xsd:element name="libros">
   <xsd:complexType>
      <xsd:choice maxOccurs="unbounded">
```

- Regla 2
  - **Cada clase** en el esquema ODL **se convierte en un elemento**, con el mismo nombre, que es incluido en el elemento *choice* creado en el paso anterior. A este elemento hay que asociarle un **tipo complejo con un elemento especial *sequence*** cuyo contenido se deriva de las siguientes reglas.

```
<xsd: element name="libro">
    <xsd:complexType>
        <xsd:sequence>
```


- Regla 3
  - Los **atributos con tipos de datos básicos** (string, short, date, float, etc) se traducen en **elementos atómicos,** con el mismo nombre, que son incluidos en la secuencia, creada en el paso anterior.

```
<xsd: element name="libro">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="isbn" type="xsd:string"/>
      <xsd:element name="titulo" type="xsd:string"/>
      <xsd:element name="año" type="xsd:unsignedshort"/>
      <xsd:element name="precio" type="xsd:float"/>
```

Bases de Datos Orientadas a Objetos
Reglas de Transformación ODL – XML Schema

- Regla 4
  - Para los **atributos key** es necesario declarar un **elemento especial key** como un sub-elemento del elemento resultado de la regla 1. Este elemento se caracteriza por
    - un atributo *name,* cuyo valor será el nombre de la clase al que pertenece la clave, más la letra K.
    - *el atributo xpath* del sub-elemento ***selector*** debe contener una expresión como *.//nombre de la clase a la que pertenece el atributo clave.*
    - *El atributo xpath del sub-elemento* **field** debe contener una expresión Xpath como: keynombre-atributo.

```
<xsd:key name="librok">
    <xsd:selector xpath=".//libro"/>
    <xsd:field xpath="isbn"/>
</xsd:key>
```


Bases de Datos Orientadas a Objetos
Reglas de Transformación ODL – XML Schema

- Regla 5 A)
  - **Cada relación** que contiene las **palabras clave** ***set, list*** **o** ***bag*** se convierte en un elemento con el mismo nombre, que es incluido en la secuencia creada en la regla 2, y como la cardinalidad es mayor que 1 se le debe asociar explícitamente el atributo *maxOccurs* con un valor *unbounded*.
  - Este elemento incluirá un **tipo de dato complejo anónimo** con un elemento especial **complexContent**, que incluirá un elemento *restriction* que contiene un atributo con el nombre del atributo clave de la clase referenciada.

```
<xsd:element name="escritopor" maxOccurs="unbounded"/>
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType">
          <xsd:attribute name="codigo" type=xsd:string">
      </xsd: restriction>
    </xsd:complextContent>
  </xsd:complexType>
</xsd:element>
```

Bases de Datos Orientadas a Objetos
Reglas de Transformación ODL – XML Schema

- Regla 5 B)
  - Se declarará un **elemento especial Keyref** como un sub-elemento del elemento que ha resultado de la regla 1. Este elemento tendrá
    - un atributo *name* con el nombre de la clase a la que pertenece la relación+Ref+nombre de la clase con la que se relaciona.
    - un atributo *refer* cuyo valor es el nombre del elemento clave de la clase referenciada
    - el atributo *xpath* del sub-elemento **selector** tendrá como valor .//nombre de la clase a la que pertenece la relación/nombre de la relación
    - el atributo *xpath* del sub-elemento **field** contendrá la expresión @nombre del atributo clave de la clase referenciada

```
<xsd:keyref name="libroRefAutor" refer="autorK">
        <xsd:selector xpath=".//libro/escritopor"/>
        <xsd:field xpath="@codigo"/>
</xsd:keyref>
```


Bases de Datos Orientadas a Objetos
Reglas de Transformación ODL – XML Schema

- Regla 6
  - Cada **relación que no contiene set, list o bag** se convierte en un elemento con el mismo nombre. La traducción es similar a la expresada en la regla anterior, con algunas excepciones:
    - el atributo *maxOccurs,* con valor *unbounded* no debería emplearse
    - en la declaración del **elemento keyref**
      - el atributo *xpath* del elemento **selector** contendrá el nombre de la clase a la que pertenece la relación
      - el atributo *xpath* del elemento **field** contendrá el nombre de la relación

```
<xsd:element name="publicadopor" type="xsd:string"/>
```

Bases de Datos Orientadas a Objetos
Reglas de Transformación ODL – XML Schema

- Regla 7
  - Cada **atributo de una clase** de tipo **list** se convierte en un **elemento,** con el mismo nombre, que es incluido en el elemento *sequence* que resulta de la regla 2. Este elemento incluye un **tipo de datos simple anónimo** que debe incluir un elemento *list con un atributo* itemType *cuyo valor puede ser definido por los usuarios o asignado por defecto.*

```
<xsd: element name="email">
   <xsd:simpleType>
            <xsd:list itemType="xsd:string"/>
   </xsd:simpleType>
</xsd:element>
```


Bases de Datos Orientadas a Objetos
XML Schema Resultante

```
<xsd:element name="bibdb">
        <xsd:complexType>
           <xsd:choice maxOccurs="unbounded">
             <xsd: element name="libro">
                    <xsd:complexType>
                       <xsd:sequence>
                          <xsd:element name="isbn" type="xsd:string"/>
                          <xsd:element name="titulo" type="xsd:string"/>
                           <xsd:element name="año" type="xsd:unsignedshort"/>
                           <xsd:element name="precio" type="xsd:float"/>
                           <xsd:element name="escritopor" maxOccurs="unbounded"/>
                               <xsd:complexType>
                                        <xsd:complexContent>
                                           <xsd:restriction base="xsd:anyType">
                                             <xsd:attribute name="codigo" type=xsd:string">
                                           </xsd: restriction>
                                        </xsd:complextContent>
                                  </xsd:complexType>
                             </xsd:element>
                          <xsd:element name="publicadopor" type="xsd:string"/>
                       </xsd:sequence>
                    </xsd:complexType>
                </xsd:element>
. .
<xsd:key name="librok">
        <xsd:selector xpath=".//libro"/>
        <xsd:field xpath="isbn"/>
</xsd:key>

<xsd:key name="autork">
        <xsd:selector xpath=".//autor"/>
        <xsd:field xpath="codigo"/>
</xsd:key>
....
<xsd:keyref name="libroRefAutor" refer="autorK">
        <xsd:selector xpath=".//libro/escritopor"/>
        <xsd:field xpath="@codigo"/>
</xsd:keyref>
```

Bases de Datos Orientadas a Objetos

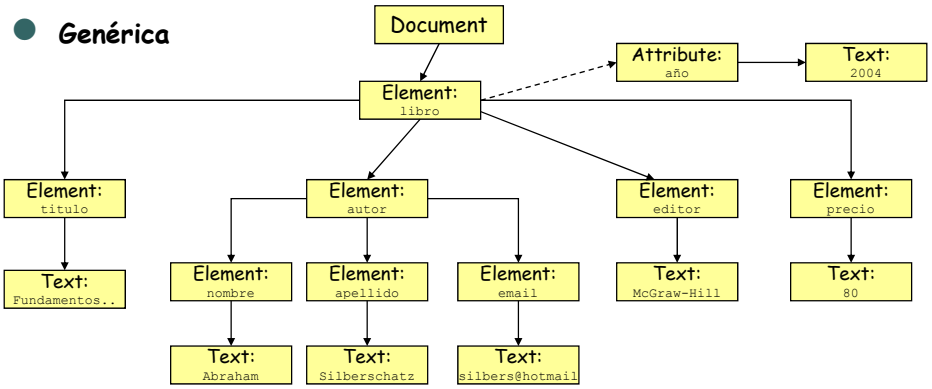
Posibilidades ofrecidas para la gestión de XML: Genérica

- **Genérica**
  - Considera toda la información del documento como objetos de clases predefinidas interconectadas mediante enlaces que preservan la estructura del documento XML
    - Ej. Únicamente hay una clase para cualquier tag XML de cualquier tipo de documento
  - DOM emplea esta aproximación y permite tratar documentos que están **bien-formados**
  - Esta opción es empleada por varias **bases de datos nativas**

Bases de Datos Orientadas a Objetos

Posibilidades ofrecidas para la gestión de XML: Genérica (II)

- **Genérica**



Bases de Datos Orientadas a Objetos

Consulta y Procesamiento

- Una vez transformado el documento XML en objetos, de forma específica o genérica, los objetos ya 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 las del propio SGBDOO, y por lo general, **no son específicos para el modelo XML**

Contenidos

- Introducción
- XML y Bases de Datos
  - **Bases de Datos Relacionales**
  - **Bases de Datos Orientadas a Objetos**
  - **Bases de Datos Nativas**
- XQuery

Bases de Datos Nativas

- **Son bases de datos**, y como tales soportan transacciones, acceso multi-usuario, lenguajes de consulta, etc., **diseñadas especialmente para almacenar documentos XML**
- Las BD nativas se caracterizan principalmente por
  - Emplear como **unidad lógica** de almacenamiento el **documento XML**
  - **Preservar el orden del documento**, las instrucciones de procesamiento, los comentarios, las secciones CDATA y las entidades
  - **Soportar lenguajes de consulta XML**
  - **No tener 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

Bases de Datos Nativas
Tipos

- **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.
    - Posibilidad sencilla: Almacenar el documento como un 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
    - Posibilidad sofisticada: Almacenar el documento en un almacén adecuado con índices, soporte para transacciones, etc.

Bases de Datos Nativas
Tipos

- **Almacenamiento Basado en el modelo**
  - Almacena un modelo binario del documento (p.e. 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

Bases de Datos Nativas
Ejemplos – Según sistema de almacenamiento

- **Sistema propietario**

Nombre	Licencia	API
Centor Interaction Server	Comercial	C++, Java
Coherty XML DB	Comercial	Java, JSP
dbXML	Comercial	Java, Javascript, JSP
eXtc	Comercial	
GoXML DB	Comercial	Java
Infonyte DB	Comercial	JSP, Java
Ipedo XML Database	Comercial	Java
Lucid XML DM	Comercial	Java
Tamino XML Server	Comercial	JSP
TeamXML	Comercial	Java
Virtuoso	Comercial	
XIndice	Open Source	Java
XDBM	Open Source	

Bases de Datos Nativas

Ejemplos – Según sistema de almacenamiento

- **Sistema relacional**

Nombre	Licencia	API
DBDOM	Open Source	Java
eXist	Open Source	Java
XDB	Open Soure	C++

- **Sistema orientados a objetos**

Nombre	Licencia	API
Birdstep RDM XML	Comercial	C, C++, Java
MindSuite XDB	Comercial	Java, C++
Ozone	Open Source	Java

Bases de Datos Nativas

Características Generales

- **Almacenamiento de documentos en colecciones**
  - Las colecciones juegan en las bases de datos nativas el papel de las tablas en las DB relacionales
  - Los documentos se suelen agrupar, en función de la información que contienen, en colecciones que a su vez pueden contener otras colecciones.
- **Validación de los documentos**
- **Consultas**
  - La mayoría de las BD XML soportan uno o más lenguajes de consulta
  - Uno de los más populares es XQuery

Bases de Datos Nativas

Características Generales (II)

- **Indexación XML**
  - Se ha de permitir la creación de índices que aceleren las consultas realizadas frecuentemente
- **Creación de identificadores únicos**
  - A cada documento XML se le asocia un identificador único por el que será reconocido dentro del repositorio
- **Actualizaciones y Borrados**
  - Las BD nativas tienen una gran variedad de estrategias para actualizar y borrar documentos
  - Muchos sistemas soportan el XUpdate para llevar a cabo esta funcionalidad

Contenidos

- Introducción
- XML y Bases de Datos
- ☞ XQuery

XQuery - Introducción

- *Query Working Group*, creado en octubre de 1999 por el W3C, es el grupo encargado de la elaboración del lenguaje de consulta XQuery para XML
- XQuery está todavía en proceso de elaboración. Actualmente es ***Recomendación Propuesta del W3C (21 de Noviembre de 2006)***
- Se han publicado diferentes borradores de trabajo:
  - **XQuery 1.0: An XML Query Language**, especifica la sintaxis y una descripción informal del lenguaje.
  - XML Query Requirements *(W3C Working Draft)*
  - XQuery 1.0 and XPath 2.0 Data Model *(W3C Working Draft)*
  - XQuery 1.0 Formal Semantics *(W3C Working Draft)*
  - XQuery 1.0 and XPath 2.0 Functions and Operators *(W3C Working Draft)*
  - XML Query Use Cases *(W3C Working Draft)*
  - Etc.

XQuery - Introducción

- Lenguaje de consulta diseñado para escribir consultas sobre colecciones de datos expresadas en XML.
  - 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
- Ha sido construido sobre la base de XPath y se fundamenta en él para realizar la selección de información y la iteración a través del conjunto de datos.

XQuery - Introducción

- Sus principales aplicaciones se resumen en tres:
  - **Recuperar información a partir de conjuntos de datos XML**. Permite filtrar los nodos que interesan de un documento XML y transformarlos para mostrar la información deseada con la estructura adecuada.
  - **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.**

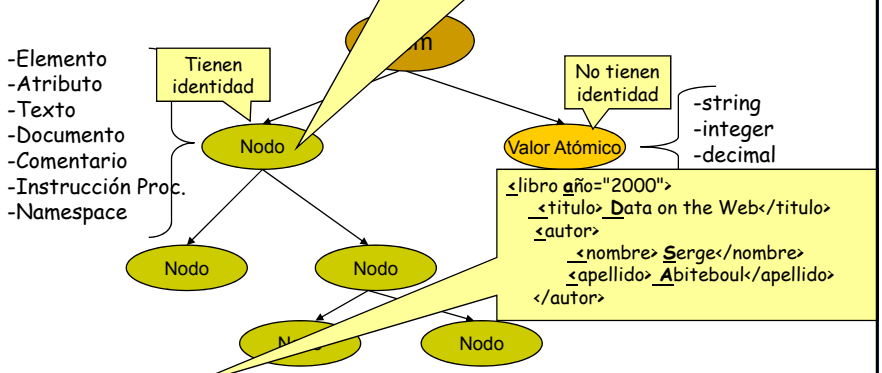
XQuery – Modelo de Datos

- La entrada y la salida de una consulta XQuery se define en términos de un modelo de datos
- El modelo de datos de la consulta proporciona una representación abstracta de uno o más documentos XML (o fragmentos de documentos)
- El modelo de datos contempla:
  - Secuencias
  - Valor especial llamado *"error value"* que es el resultado de evaluar una expresión que contiene un error

XQuery – Modelo Características

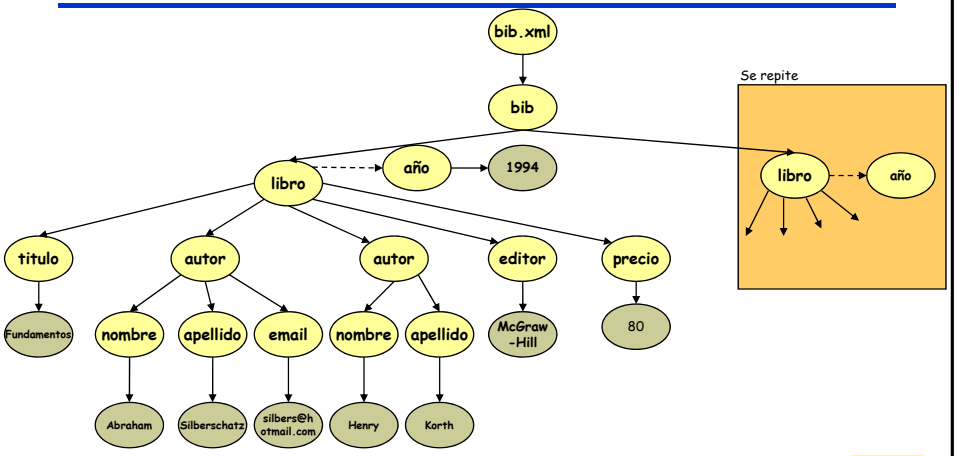
Cada nodo tiene:
-Identidad
-Valor tipado (secuencia de cero o más valores atómicos)
-Valor string (valor de tipo xs:string)
-Nombre (no siempre)

- Se basa en la noción de **secuencia:** Col[illegible]nada de cero o más ítems

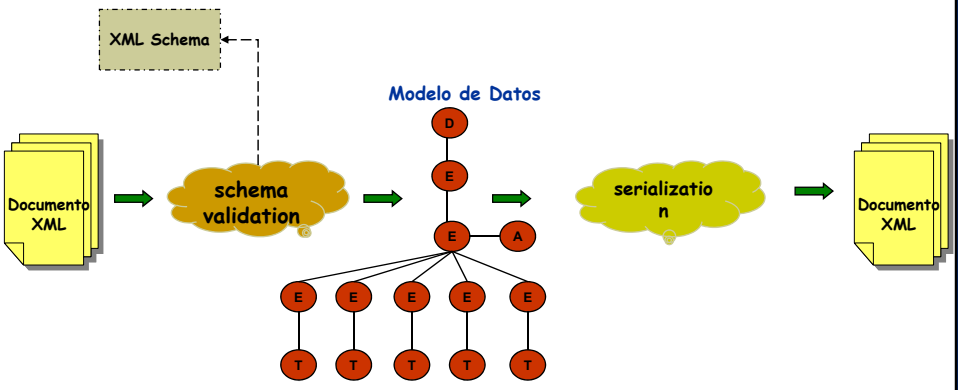


- **Orden del documento:** corresponde al orden en que los nodos aparecerían si la jerarquía de nodos fuese representada en formato XML (si el primer carácter de un nodo ocurre antes que el primer carácter de otro nodo, lo precederá también en el orden del documento)

XQuery – Modelo de Datos – Ejemplo



XQuery – Modelo de Datos – Transformación



XQuery – Comentarios

- Xquery es un lenguaje sensible a mayúsculas. Va todo en minúsculas
- Los comentarios se expresan entre (: :)
  - (: Esto es un ejemplo de comentario :)

XQuery – Expresiones

- XQuery presenta varios tipos de expresiones
- El valor de una expresión es una secuencia heterogénea de nodos y valores atómicos
- La mayoría de las expresiones son compuestas por expresiones de más bajo nivel combinadas mediante operadores y palabras reservadas

XQuery – Valores Atómicos

- Namespace denominación: **XML Schema Types**
- Namespace prefijo : **xs**
- Namespace URI: http://www.w3.org/2001/XMLSchema

- Expresiones constituidas por Valores Atómicos
  - **53** -> literal integer (xs:integer)
  - **5.3** -> literal decimal (xs:decimal)
  - **5.3E3** -> literal double (xs:double)
  - **"57"** -> literal string
  - **'57'** -> literal string
  - **date ("2004-3-30")** -> invocación del constructor
  - **(2 + 4) * 5** -> los paréntesis expresan el orden de evaluación
  - **(1, 2, 3)** -> operador , concatena valores para formar una secuencia
  - **1 to 3** -> devuelve la secuencia 1, 2, 3
  - **$inicio** -> representa una variable que puede ser utilizada en una expresión
  - **substring("Curso Extension", 1, 5)** -> invocación de funciones

- La instrucción **let**
  let $inicio :=1 , $fin:=15
  asocia un valor a una variable para ser empleada en una expresión

• Namespace denominación: **Built-in functions**
• Namespace prefijo : **fn**
• Namespace URI: http://www.w3.org/2005/xpath-functions

XQuery – Funciones de Entrada

- Ejemplo de invocación de funciones para acceder a los datos de entrada
  - **fn:doc(URI).** Devuelve el nodo raíz del documento referenciado por un identificador URI. Es la función más habitual para recuperar datos de archivos.

fn:doc ("bib.xml")

(:Devuelve el documento completo, el nodo documento :)
(:El documento se identifica por su URI,Universal Resource Identifier:)

  - **fn:collection().** Devuelve una secuencia de nodos referenciados por una URI, sin necesidad de que exista un nodo documento o raíz. Es la función más habitual para acceder a la información almacenada en una base de datos que tenga capacidad para crear estructuras de datos XML.

fn:collection ("empleados")

*Obtener todos elementos **libro** contenidos en el documento*

document ("bib.xml")//libro

XQuery – Path Express

- Expresiones constituidas por Path Expressions para localiza s:
  - Path Expression en XQuery están basadas en la sintáxis de XPath
    - Está constituida por una serie de pasos separados por "/" or "//"
    - El resultado de cada paso es una secuencia de nodos
    - Su valor es la secuencia de nodos que resultan del último paso en el *path*
- Ejemplo:
  - *Obtener todos los elementos "libro" hijos del elemento "bib" en el documento "bib.xml"*

doc("bib.xml")/bib/libro

/bib

/libro

Devuelve el nodo documento

Selecciona el elemento **bib** en la cima del documento

Selecciona los elementos **libro** hijos del elemento **bib**

XQuery – Path Expres

• Los nodos seleccionados por cada paso sirven como **nodos de contexto** para los siguientes pasos
•Si un paso tiene varios nodos de contexto se evalua cada uno de ellos y el resultado es la **union** de los resultados dela evaluación de cada uno de ellos
•El resultado es siempre una secuencia de **nodos distintos** (no se admiten duplicados)

- Evaluación de las Path Expressions:
  - Cada paso es evaluado en el contexto de un nodo concreto (**nodo de contexto**)
  - Un paso puede ser cualquier expresión que devuelva una secuencia de nodos
  - Un **paso eje** comienza en el nodo de contexto y se desplaza a través de la jerarquía de nodos en una dirección concreta (la del eje)
  - Los criterios de selección para los nodos son:
    - **Nombre**
    - **Posición con respecto al nodo de contexto**
    - **Predicado basado en el valor de un nodo**

- **Path Expressions** pueden ser escritas en su versión abreviada o **no abreviada**

XQuery – Expresiones - Predicados

- Un predicado es una expresión encerrada entre corchetes que se emplea para filtrar una secuencia de valores
  - Ej1. libro [titulo = " mi titulo"]
  - Ej2. libro [precio >10]
  - Ej3. libro[5] ->selecciona el quinto nodo hijo
  - Ej4. libro[precio]->selecciona los libros que tienen un nodo hijo precio
  - Ej.5 libro[@año>1997] -> selecciona los libros cuyo año es posterior al 97

XQuery – Ejemplos

- 1. Devolver los títulos de todos los libros publicados después de 1997

```
doc ("bib.xml")/bib/libro[@año>1997]/titulo
```

- 2. Devolver el primer autor de cada libro

```
doc("bib.xml")/bib/libro/autor[1]
```

- 3. Devolver los apellidos de los autores de los libros de bib.xml

```
doc ("bib.xml")/bib/libro/autor/apellido
```

- 4. Devolver los autores que tienen correo electrónico

```
doc ("bib.xml")/bib/libro/autor[email]
```

XQuery – Ejemplos

- 5. Obtener el primer autor que aparece en el documento

```
(doc("bib.xml")/bib/libro/autor)[1]
```

- 6. Obtener los años en los que fueron publicados los libros que aparecen en bib

```
doc("bib.xml")/bib/libro/@año
```

- 7. Obtener los libros cuyos autores se apellidan Korth

```
doc("bib.xml")/bib/libro/autor[apellido="Korth"]
```

XQuery – Creación de nodos

document{ } crea un nodo documento vacio

- Xquery proporciona un constructor de nodos documento:
- document( ) crea un nodo documento vacío.

```
8. document{
  <?xml-stylesheet type="text/xsl" href="c:\temp\ejemplo.xslt" ?>,
      <libro>
        <titulo> Fundamentos de Bases de Datos </titulo>
        . . .
      </libro>
}
```

- Los constructores pueden combinarse con otras expresiones XQuery para generar contenido dinámicamente
  - En un elemento constructor los { } delimitan las expresiones que son evaluadas para crear un documento abierto

XQuery – Creación de nodos

```
<ejemplo>
    <p> Esto es una consulta de ejemplo </p>
    <ej> doc("bib.xml")//libro[1]/titulo </ej>
    <p> Este es el resultado de la consulta anterior </p>
    <ej> {doc("bib.xml")//libro[1]/titulo } </ej>
</ejemplo>
```

Salida

```
<ejemplo>
    <p> Esto es una consulta de ejemplo </p>
    <ej> doc("bib.xml")//libro[1]/titulo </ej>
    <p> Este es el resultado de la consulta anterior </p>
    <ej> Fundamentos de Bases de Datos </ej>
</ejemplo>
```

XQuery – Creación de nodos – Ejemplos

- 9. Mostrar los títulos de los libros del fichero bib de la forma siguiente:

```
<titulos num="5">
   <titulo>Fundamentos de Bases de Datos</titulo>
   <titulo>Advanced Programming in the UNIX Environment</titulo>
   <titulo>Data on the Web</titulo>
   <titulo>The XML Handbook</titulo>
   <titulo>The XML Handbook 2</titulo>
</titulos>
```

Solución:

```
<titulos num= "{ count (doc("bib.xml")//titulo) }">
        {doc("bib.xml")//titulo}
</titulos>
```


XQuery – Expresiones - Constructores

- Constructores calculados de elementos y atributos
  - Representan una sintaxis alternativa
  - Ejemplo:

```
element libro{
        attribute año {2001},
        element titulo {"Usabilidad de Sitios Web"},
            element autor{
                element nombre {"Jakob"},
                element apellido {"Nielsen"}
            },
            element editorial {"Addison-Wesley"},
            element precio {45.78}
    }
```

XQuery – Expresiones FLWOR

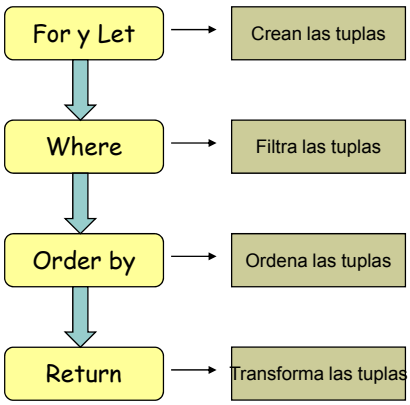
- Permiten la combinación y re-estructuración de los nodos
- FLWOR (For Let Where Order Return) es una de las expresiones más potentes y típicas de XQuery
- Se basa en ligar valores a variables con las cláusulas for y let y emplear esas variables para crear nuevos resultados
- Ejemplo:

- 10. *Devolver el título de los libros que fueron publicados en el año 2000*

```
for $b in doc("bib.xml")//libro
  where $b/@año="2000"
  return $b/titulo
```

XQuery – Expresiones FLWOR (II)

- Una expresión FLWOR
  - Comienza con una o más cláusulas *for* o *let* en cualquier orden (al menos uno de ellas)
  - Seguidas por una cláusula *where* opcional
  - Puede aparecer una cláusula *order by* opcional
  - Finaliza con una cláusula *return* obligotoria



* tupla= cada uno de los valores que toma una variable

XQuery – Cláusula for

- **Cláusula for**
  - Proporciona una manera de iterar sobre una secuencia de valores, ligando una variable a cada uno de los valores y evaluando una expresión para cada valor de la variable
  - Ejemplos:

Secuencia de valores Expresión a evaluar

1) for $n in (2,3) return $n+1 Resultado → (3,4)

2) for $m in (2,3), $n in (2,5)
return <fact> $m veces $n es ($m*$n) </fact>

```
<fact> 2 veces 2 es 4 </fact>
<fact> 2 veces 5 es 10 </fact>
<fact> 3 veces 2 es 6 </fact>
<fact> 3 veces 5 es 15 </fact>
```

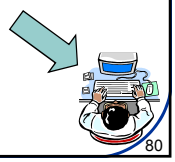
XQuery – Cláusula let

- **Cláusula let**
  - Liga variables al resultado entero de una expresión y devuelve una única tupla
  - Ejemplo:

for $i in (1 to 3) -> liga i a cada ítem en la secuencia 1 to 3

let $j := (1 to $i) -> liga i a toda la secuencia (i to $i) Resultado →

```
$i =1, $j =1
$i =2, $j = (1,2)
$i =3, $j = (1,2,3)
```



XQuery – Cláusula where

- **Cláusula where**
  - Filtra las tuplas producidas por la cláusulas let y for.
  - Contiene una expresión que es evaluada para cada tupla. Si su evaluación es false esa tupla es descartada

XQuery – Cláusula order by

- **Cláusula order by**
  - Esta cláusula es ejecutada una vez para cada tupla retenida por la cláusula where
  - Esta cláusula es evaluada antes que la cláusula return
  - Los resultados de estas ejecuciones son concatenados en una secuencia que sirve como resultado de la expresión FLWOR

```
for $i in doc("bib.xml")//titulo
order by ($i)
return $i
```

XQuery – Cláusula order by. Ejemplos

- 11. Obtener los autores ordenados ascendentemente por apellido y descendentemente por nombre.

```
for $a in doc("bib.xml")//autor
  order by $a/apellido ascending, $a/nombre descending
  return $a
```

- 12. Obtener los títulos de los libros ordenados por el nombre del primer autor

```
for $a in doc("bib.xml")//libro
 order by $a/autor[1]/nombre
   return $a/titulo
```


XQuery – Cláusula return

- **Cláusula return**
  - Esta cláusula es ejecutada una vez para cada tupla retenida por la cláusula where
  - Los resultados de estas ejecuciones son concatenados en una secuencia que sirve como resultado de la expresión FLWOR
  - Suelen contener constructores de elementos para cambiar la jerarquía de datos

```
for $b in doc("bib.xml")//libro
 return
   <autor>  {string($b/nombre), " ", string($b/apellido)} </autor>
```

XQuery – Variable posicional *at*

- **Variable posicional *at***
  - La claúsula for soporta variables posicionales que identifican la posición de un elemento en la expresión que es generada.

```
for $b at $i in doc("bib.xml")//titulo
  return
    <titulo pos="$i"> {$b} </titulo>
```


XQuery – Ejemplos FLWOR

- 13. Listar el título y precio de los libros cuyos precios sean inferiores a 100.0 euros. Sacar a continuación solo los que tengan precio

```
for $b in doc("bib.xml")//libro
where $b/precio < 100
return <informe> {$b/titulo, $b/precio } </informe>
```

- 14. Listar el título de cada libro junto con el número de autores

```
for $b in doc("bib.xml")//libro
let $c := $b/autor
return <libro> { $b/titulo, <numero> {count ($c)} </numero> }</libro>
```

XQuery – Ejemplos

- 15. Listar los títulos de los libros con más de dos autores

```
for $b in doc("bib.xml")//libro
let $c := $b/autor
where count ($c) >2
return <autores> {$b/titulo, <numero>{count($c)}</numero>} </autores>
```

- 16. *Listar los títulos de los libros así como cualquier revisión que haya sobre ellos*

```
for $b in doc("bib.xml")//titulo
for $i in doc("revisores.xml")//revision
where $i/titulo = $b
return <revision>{$b, $i/comentarios} </revision>
```


XQuery – Ejemplos

- *17. Obtener el título y el año de todos los libros publicados después de 1991.*

```
for $i in doc("bib.xml")//libro
where $i/@año >1991
return <libro año="{$i/@año}"> {$i/titulo} </libro>
```

- *18. Obtener el título y el año de todos los libros publicados por la editorial Addison-Wesley después de 1991*

```
for $i in doc("bib.xml")//libro
where $i/@año >1991 and contains ($i/editorial,"Addison-Wesley")
return <libro año="{$i/@año}"> {$i/titulo, $i/editorial} </libro>
```

XQuery – Ejemplos

- *19. Obtener los títulos de los libros y precio para aquellos que tengan unos precios inferiores en amazon que en bib.*

<libros-precios-baratos-amazon>
<titulo>Fundamentos de Bases de Datos</titulo>
<precio-amazon>75</precio-amazon>
<precio-bib>80</precio-bib>
</libros-precios-baratos-amazon>

```
for $i in doc("bib.xml")//libro
  for $j in doc("amazon.xml")//libro
    where $i/titulo = $j/titulo and $i/precio>=$j/precio
      return
        <libros-precios-baratos-amazon>
       {
          $i/titulo,
        <precio-amazon>{ string($j/precio) }</precio-amazon>,
        <precio-bib> { string($i/precio) }</precio-bib>
       }
    </libros-precios-baratos-amazon>
```


XQuery – Expresiones condicionales

- if (expresion) then

  else (se ejecuta si la expresión devuelve false o una secuencia vacia).
  - La claúsula else es obligatoria y debe aparecer siempre en la expresión condicional. Si no hay instrucciones para añadirle se puede indicar como else().

- Ejemplo:

```
if (count($b/autor)>2)
 then <autor> et al. </autor>
else ()
```

XQuery – Ejemplos

- *20. Listar todos los libros de bib.xml y sus dos primeros autores. Si el libro tiene más de dos autores se añadirá un tercer autor 'et al'.*

```
<lista-libros>
  <libro>
    <titulo>Data on the Web</titulo>
    <autor>Serge , Abiteboul</autor>
    <autor>Peter , Buneman</autor>
    <autor>et. al</autor>
  </libro>
```

```
<lista-libros>{
for  $a in fn:doc("bib.xml")//libro
return
  <libro>
  {$a/titulo}
    {     for $b at $i in $a/autor
            where $i<=2
            return <autor> {string($b/nombre),",",string($b/apellido)} </autor>    }
   {
     if (count($a/autor) > 2 )
       then <autor> et. al </autor>
     else ( )       }
 </libro>      }
</lista-libros>
```


XQuery – Expresiones Cuantificadas

- Cuantificador existencial (*some*)
  - Comprueba si una condición es cierta para algún valor en la secuencia
  - Ejemplo:

```
some $n in (5, 7, 9, 11) satisfies $n > 10
```

Para cada valor se evalua la expresión de test

Devuelve true porque la expresión de test es cierta para algunos valores

- Cuantificador universal (*every*)
  - Comprueba si una condición es cierta para cada valor en la secuencia
  - Ejemplo:

```
every $n in (5, 7, 9, 11) satisfies $n > 10
```

Devuelve false porque la expresión no es cierta para todos los valores

XQuery – Ejemplos Cuantificadores Existenciales

- 21. Listar los títulos de los libros en los que al menos uno de sus autores es Silberschatz.

```
for $b in doc("bib.xml")//libro
  where some $a in $b/autor satisfies ($a/apellido="Silberschatz")
  return $b/titulo
```

- 22. Listar los títulos de los libros en los que todos sus autores sean Silberschatz

```
for $b in doc("bib.xml")//libro
  where every $a in $b/autor satisfies ($a/apellido="Silberschatz"))
  return $b/titulo
```

XQuery – Operadores

- **Comparación de valores:** eq, ne, lt,le, gt, ge
  - Comparan dos valores escalares y produce un error si alguno de los operandos es una secuencia de longitud mayor de 1
- **Comparación generales:** =, !=, >, >=, <, <=
  - Permiten comparar operandos que sean secuencias
- **Comparación de nodos:** is e is not
  - Comparan la identidad de dos nodos. Ej. $nodo1 is $nodo2 es true si ambas variables están ligadas al mismo nodo
- **Comparación de posición de los nodos:** <<
  - Compara la posición de dos nodos. $node1<<$node2 es true si el nodo ligado a $node1 ocurre primero en el orden del documento que el nodo ligado a $nodo2
- **Lógicos:** and y or
  - Se emplean para combinar condiciones lógicas dentro de un predicado. Ej. Item[seller="Smith" and precio]
- **Sobre secuencias de nodos:** union, intersect y except
  - Devuelven secuencias de nodos en el orden del documento y eliminan duplicados de las secuencias resultado
- **Aritméticos:** +, -, *, div, idiv,mod
  - Son definidos sobre valores numéricos
- **Negación:** not
  - Es una función más que un operador. Invierte un valor booleano.

XQuery – Ejemplos Operadores

- **Unión (union).** Recibe dos secuencias de nodos como operandos y devuelve una secuencia de nodos con la suma de los nodos de las dos secuencias originales.
- 23. Listar ordenadamente los apellidos de todos los autores y editores

```
for $i in distinct-values(doc("bib.xml")//(autor union editor) /apellido)
 order by $i
   return <apellido> {$i} </apellido>
```

- **Intersección (intersect).** Recibe dos secuencias de nodos como operandos y devuelve una secuencia que contiene todos los nodos que aparecen en ambos operandos.
- **Diferencia (except).** Recibe dos secuencias de nodos como operandos y devuelve una secuencia conteniendo todos los nodos del primer operando que no aparezcan en el segundo operando.

XQuery – Funciones Predefinidas

- **Funciones de entrada:** doc() y collection()
- **Funciones agregadas:** **sum, avg,count, max, min,** que operan sobre una secuencia de números y devuelven un resultado numérico
- **Funciones de cadena:** string-length(), substring(), upper-case(), lower-case(), concat(), string(), starts-with(), ends-with(),etc.
- **Funciones generales:** distinct-values(), empty(), exits(), …
- Etc.

XQuery – Ejemplos Funciones Predefinidas

- 24. Listar todos los apellidos distintos de los autores.

```
for $i in distinct-values(doc("bib.xml")//autor/apellido)
 return <apellidos> {$i} </apellidos>
```

- 25. Listar todos los libros que tengan al menos un autor

```
for $i in doc("bib.xml")//libro
where not (empty($i/autor))
return $i
```

```
for $i in doc("bib.xml")//libro
where exists ($i/autor)
return $i
```


XQuery – Ejemplos Funciones Predefinidas

- 26. Obtener el promedio de autores por libro

<promedio-de-autores>
1.75
<promedio-de-autores>

```
<promedio-de-autores>
{
  let $a := doc("bib.xml")//libro
   return
      avg(for $as in $a return count($as/autor))
 }
</promedio-de-autores>
```

- 27. Mostrar los libros cuyos precios superan la media

```
let $l := doc("bib.xml")//libro
  let $c:= avg($l//precio)
    return  $l[precio>$c]
```

XQuery – Funciones Definidas por el Usuario

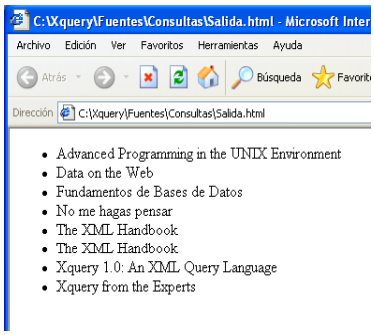
- Definidas por el usuario:

```
declare function local:libros-por-autor($nombre,$apellido) as element()*
{
   for $b in doc("bib.xml")//libro
     where some $a in $b/autor satisfies ($a/apellido=$apellido and
                                          $a/nombre=$nombre)
     order by $b/titulo
     return $b/titulo
};

<ejemplo-funcion>{

for $i in doc("bib.xml")//autor
  return local:libros-por-autor($i/nombre, $i/apellido)
}
</ejemplo-funcion>
```


XQuery – Ejemplos

- 28. Listar los títulos de los libros de esta manera:

```
<html>
  <ul>
    {
      for $x in doc("bib.xml")//titulo
      order by $x
      return <li>{$x}</li>
    }
</ul>
</html>
```



XQuery – Ejemplos

- 29. Crear una tabla HTML con los títulos de todos los libros del documento bib.xml

```
<html>
   <head> <title> Titulos de libros seleccionados </title>
  </head>
  <body>
      <table>{
        for $b in doc("bib.xml")//libro
            return
           <tr> <td> <i> {string($b/titulo)} </i> </td> </tr>}
    </table>
  </body>
</html>
```


XQueryX

- **XQueryX**
  - XQuery debe permitir utilizar diferentes sintaxis para realizar las consultas.
  - XQueryX es un ejemplo de sintaxis alternativa para Xquery que permite definir una consulta Xquery mediante etiquetas XML.
    - *Recomendación Candidata (Enero 2007)*
    - Una sintaxis basada en XML para Xquery facilita el uso de herramientas estándares y generar y consultar los contenidos de la consulta.

XQueryX

```
<bib>
      {
          for $b in doc("bib.xml")/bib/libro
           where $b/editorial = "Addison-Wesley" and $b/@año > 1991
             return
                    <libro año="{$b/@año }">
                           { $b/titulo }
                    </libro>
         }
</bib>
```

Código XQueryX equivalente

XQuery y Java

- **XQEngine**
  - Motor GPL para **Xquery** escrito en Java.
  - Pequeño tamaño lo que facilita la integración en las aplicaciones
  - Se distribuye con una aplicación de ejemplo sencilla "SampleApp.java":
    - Indica como se puede emplear este motor en el desarrollo de una aplicación
    - Es a su vez una interfaz sencilla para ejecutar consultas Xquery.
  - Ejemplo de uso y de incrustación en una aplicación Java.

XQuery y Java

- **XQJ (XQUERY API for Java)**
  - Actualmente es un ***Early Draft Review 2 (June 2006)***
  - Permite:
    - A un programa Java conectarse a diferentes orígenes de datos
    - Preparar y ejecutar consultas con **XQuery**
  - Ejemplo de implementaciones: DataDirect XQuery (Xquery y XQJ para Oracle, SQL-Server, DB2, etc.).

XQuery y Java

- **XQJ (XQUERY API for Java)**

1. Crear una conexión:
```
QXConnection conn  = xqds.getConnection();
```
2. Crear una expresión (para ejecutar la consulta)
```
XQExpression expr = conn.createExpression();
```
3. Especificar la consulta a ejecutar
```
String c = "for $n in fn:doc("bib.xml")//libro+ "return fn:data($n/titulo)";
```
4. Ejecutar la expresión Xquery
```
XQResultSecuence result = expr.executeQuery(c);
```
5. Procesar el resultado de la secuencia iterativamente
```
while (result.next()) {
    String str = result.getAtomicValue();
    System.out.println("Título = "+str);  }
```
6. Liberar recursos asignados
```
result.close();
expr.close();
conn.close();
```

Motores XQuery Open-Source

- **Qexo**
  - Motor Xquery escrito en Java, se ejecuta sobre la plataforma Java (compilando expresiones Xquery a bytecode).
  - Tiene licencia GPL
  - Se distribuye integrado dentro del paquete Kawa

Motores XQuery

- **Qexo. Ejemplo**

1) Crear el programa Xquery

Ver código de EjemploQexo.xql.

2) Compilar el fichero Xquery a servlet

- `java -jar kawa-1.7.jar --xquery --servlet -C EjemploQexo.xql`

3) Configurar el Servidor Web

- **Necesario un servidor web que soporte servlets** (ej. Tomcat)
- **Instalar en el servidor un fichero .jar de Kawa** (kawa-1.7.jar)
- **Copiar los .class necesarios en el directorio adecuado del servidor**
- **Añadir en el fichero de configuración *web.xml* la siguiente información**

```
<web-app>
  <display-name> Ejemplo Xquery </display-name>

 <servlet>
    <servlet-name> EjemploQexo </servlet-name>
    <servlet-class>gnu.kawa.servlet.EjemploQexo </servlet-class>
 </servlet>


 <servlet-mapping>
          <servlet-name> EjemploQexo </servlet-name>
          <url-pattern> *.xql </url-pattern>
 <servlet-mapping>
</web-app>
```

Motores XQuery Open-Source (II)

- **Saxon**
  - Motor Xquery escrito en Java.
  - Se distribuye en dos paquetes:
    - **Saxon-B.** Open-Source. Contiene una implementación básica de XSLT 2.0 y de Xquery
    - **Saxon-A.** Software propietario. Licencia de evaluación 30 días. Contiene procesador completo de XSLT y Xquery.

Motores XQuery Open-Source (III)

- **Quizx/open**
  - Motor Xquery escrito en Java.
  - Tiene licencia GPL
  - Implementa todas las características del lenguaje excepto la importación y la validación de XML Schemas.

Herramientas Relacionadas

- **XQuare Fusion y XQuare Bridge** (antes Xquark). Open Source
  - **Xquare Fusion.**
    - Motor que permite consultar en tiempo real diferentes orígenes de datos basándose en XML y en el lenguaje XQuery para obtener los documentos.
  - **Xquare Bridge.**
    - Permite importar y exportar datos a bases de datos relacionales empleando XML. Incluye dos módulos
      - *Extractor,* extrae los datos almacenados en bases de datos relacionales a XML por medio de XQuery
      - *Mapper.* Traduce documentos XML y los almacena en bases de datos relacionales existentes.
    - Soporta XQuery
    - Soporta las bases de datos:MySQL, Oracle, MS SQLServer, Sybase

Herramientas Relacionadas (II)

- **BumbleBee**
  - Entorno de prueba automático **creado con el fin de evaluar** motores Xquery y validar consultas escritas en dicho lenguaje.
  - Permite evaluar el grado de satisfacción de los borradores del estándar con motores XQuery.
  - Distribuye un conjunto de pruebas ya preparadas y, además ofrece un entorno sencillo para redactar y ejecutar nuestras propias pruebas.
  - Es software propietario aunque está disponible para descarga una versión de demostración completamente funcional durante 30 días.

Direcciones de interés

XQuery

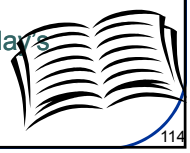
- W3C XML Query
  http://www.w3.org/XML/Query/
- XQuery 1.0: An XML Query Language
  http://www.w3.org/TR/xquery/
- XML Syntax for XQuery 1.0 (XQueryX)
  http://www.w3.org/TR/2003/WD-xqueryx-20031219/
- JSR-000225 XQuery API for JavaTM (XQJ)
  - **http://jcp.org/aboutJava/communityprocess/edr/jsr225/**

Bibliografía

📖 Xquery from the Experts. A Guide to the W3C XML Query Language
Chamberlin D., Draper D., et. al
Addison-Wesley, 2005.

📖 Xquery 1.0: An XML Query Language
W3C Candidate Recommendation, June 2006
http://www.w3.org/XML/Query

📖 Data on the Web. From relations to semistructured data and XML
Abiteboul S., Buneman P. Y Suciu Dan
Morgan Kaufmann, 2000

📖 The Object Database Standard: ODMG 3.0.
Rick Cattell et al.
Morgan Kaufmann, 1999.

📖 Succeeding with Object Databases: A practical look at today's implementations with Java and XML.
A. Chaudhri y R. Zicari et al.
John Wiley & Sons, 2000.