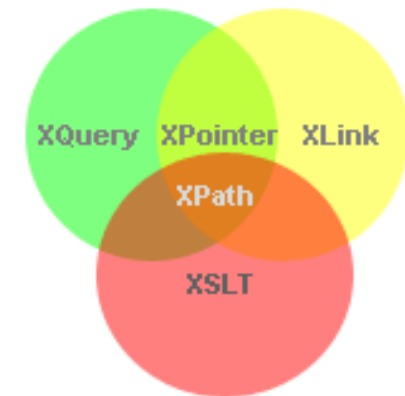


XQuery

REPOSITARIOS DE INFORMACION

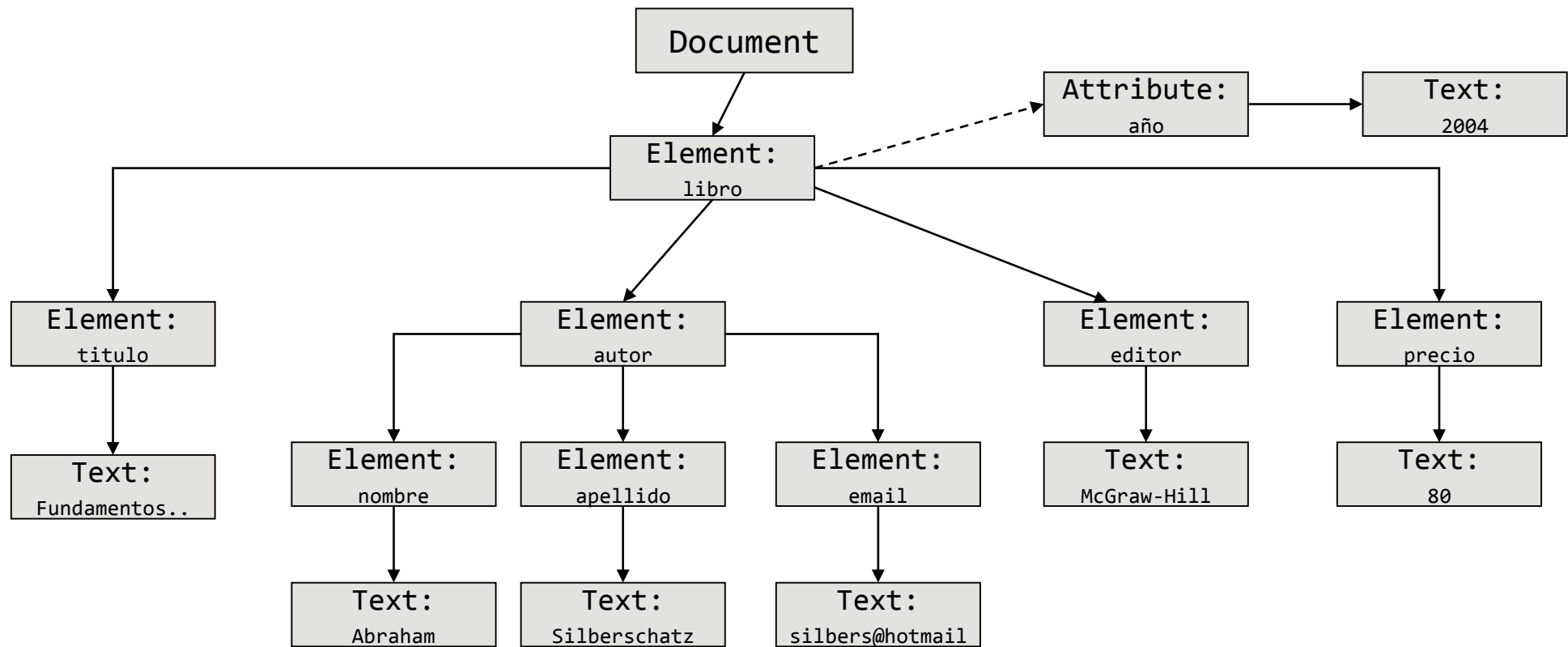
Grado en Ingeniería Informática del Software

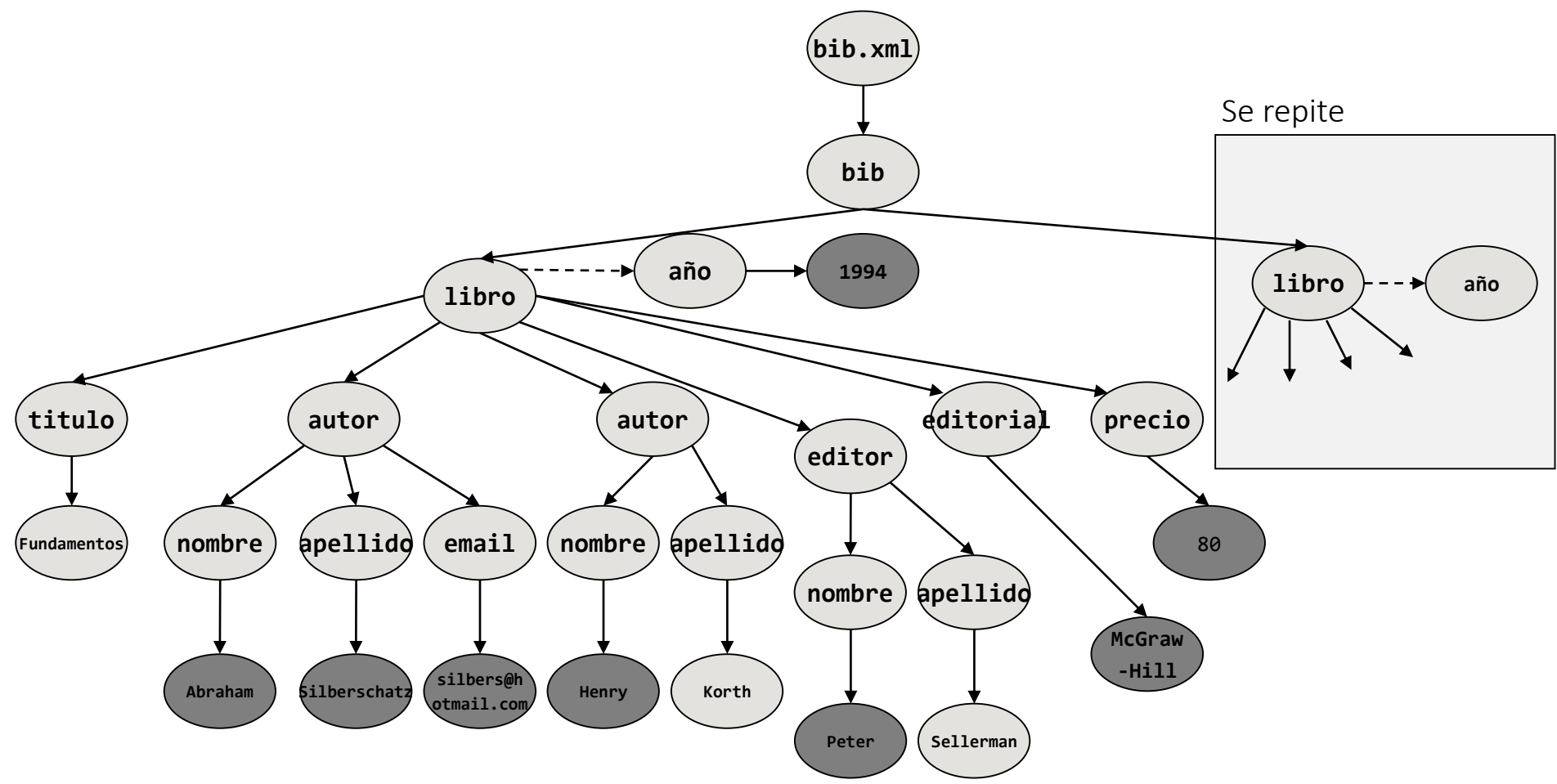
- *Query Working Group*, creado en octubre de 1999 por el W3C, es el grupo encargado de la elaboración del lenguaje de consulta **XQuery para XML**
- **XQuery** surge de la necesidad de disponer de un lenguaje estándar de consultas para acceder a la creciente cantidad de información almacenada en formato XML
- Es una importante extensión de **XPath 2.0**
- La segunda edición de ***XQuery 1.0: An XML Query Language*** fue publicada como **recomendación en diciembre de 2010**.
 - <http://www.w3.org/TR/xquery/>



- XPath y XQuery son estándares para acceder y obtener datos desde un documento XML
- **XPath** es un lenguaje para referirse a caminos dentro de un XML (caminos en el sentido del árbol XML visto como un grafo)
- **XQuery** es un lenguaje más completo con funcionalidades similares a SQL

- Lenguaje pensado para la selección de elementos de un documento XML para su procesamiento
- Un procesador de lenguaje XPath toma como entrada un árbol de nodos del documento origen, selecciona una parte de él y genera un nuevo árbol que puede ser tratado
- El **modelo de datos** es la representación que XPath tiene de un documento XML. Para XPath un documento XML es un **árbol de nodos**





- Sirven para seleccionar elementos que se encuentran en cierto camino en el árbol
- Los descriptores se forman simplemente nombrando *tags* separados por /
- Si el descriptor comienza con / se supone que es un camino desde la raíz
- Si el descriptor comienza con // se supone que el camino descrito puede comenzar en cualquier parte del árbol
- Se supone que todos los descriptores se refieren a caminos que avanzan en la profundidad del árbol

- **/bib/libro/autor/nombre**

- Selecciona la lista de todos los tag *nombre* de todos los *autores* de todos los libros en *bib*

- **//nombre**

- Selecciona la lista de todos los tag *nombre* en cualquier lugar del documento

- **//autor/nombre**

- Selecciona la lista de todos los tag *nombre* que son hijos de un tag *autor* en el árbol partiendo desde cualquier punto

- Siguiendo el modelo del ejemplo anterior el descriptor **/bib/libro/editor**, ¿Qué devolverá?

- El **operador *** se usa como comodín para nombrar a cualquier tag
- El descriptor **/*/*/*nombre*** selecciona la lista de todos los tag *nombre* que se encuentran a dos niveles de profundidad en el árbol
- El descriptor **/*/*/*/*email*** selecciona la lista de todos los tag *email* que se encuentran a tres niveles de profundidad en el árbol

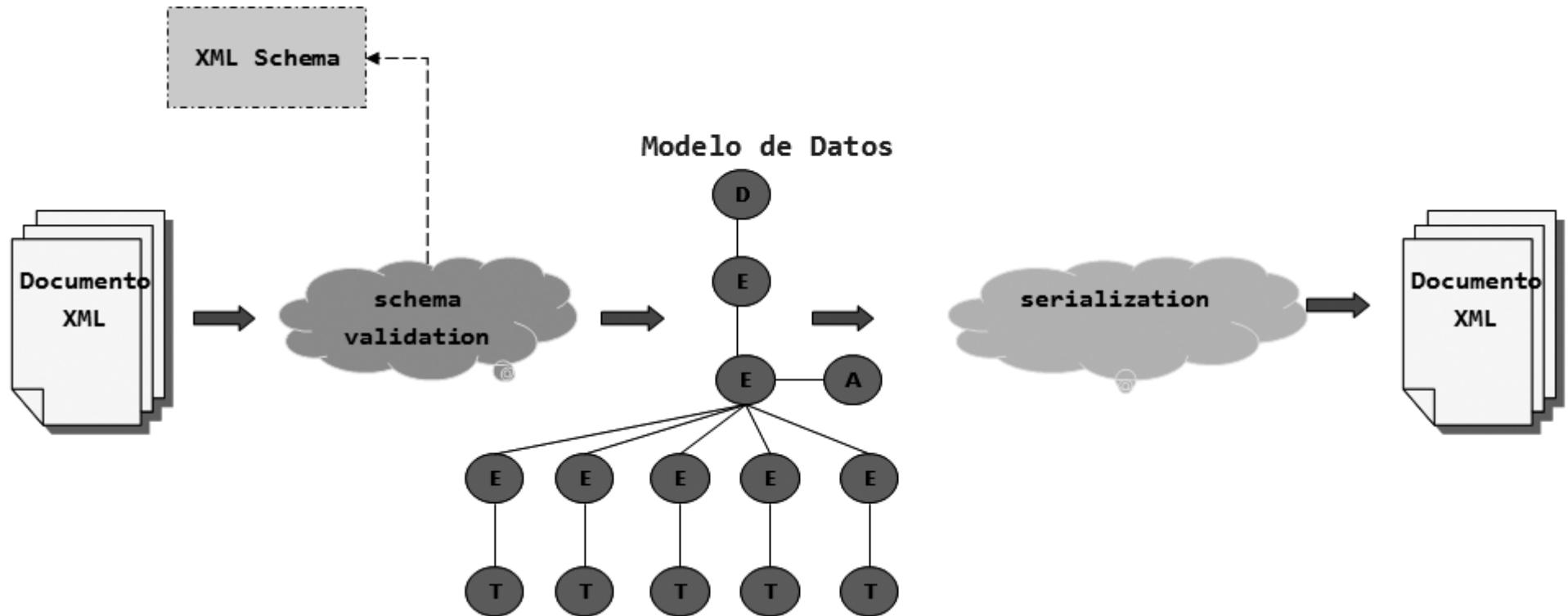
- **@** se utiliza para referirse a los atributos de los elementos.
 - Ejemplo:
 - @año
- En un descriptor de camino los atributos se nombran como si fuesen tag hijos pero anteponiendo @
 - El descriptor **/bib/libro/@año** selecciona los años de los libros que están en bib.

- En un descriptor una condición entre paréntesis cuadrados [] puede seguir al nombre de un tag o atributo (sin emplear /)
- En este caso se seleccionan los elementos que siguen el camino del descriptor pero que además cumplen la condición especificada
- En las condiciones se pueden usar comparadores (>, <, =, >=, <=, !=) y conectivos lógicos (**or** y **and**).
 - **/bib/libro[@año>1990]**, selecciona la lista de libros cuyo año de publicación es posterior a 1990
 - **//libro[titulo="Fundamentos"]/autor/nombre**, selecciona la lista de nombres de autores que han escrito libros cuyo título es Fundamentos donde quiera que estén ubicados (desde el tag libro)

- En los predicados también se pueden utilizar:
 - Funciones predefinidas (**last**, **count**, **round**, **sum**, **contains**, **string-length**, etc.)
 - Expresiones matemáticas (**+**, **-**, **div**, etc.)
- Ejemplos:
 - **doc("bib.xml")/bib/libro[1]/autor/nombre**, selecciona los nombres de los autores del primer libro que aparece en bib.
 - **doc("bib.xml")/bib/libro[1]/autor[last()]/nombre**, selecciona el nombre del último autor del primer libro que aparece en bib
 - **doc("bib.xml")/bib/libro[count(autor)=1]/titulo**, selecciona la lista de los títulos de los libros que solo tienen un autor
 - **/bib/libro[contains(titulo,"Fundamentos")]/round(precio*1.21)**, devuelve el precio de los libros que contienen la palabra "Fundamentos" incrementando en un 21% su valor (por el IVA) y redondeado el resultado

- 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

- 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 es compartido con XPath 2.0



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

- 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

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

Funciones de Entrada

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

- Ejemplo de invocación de función 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")
```

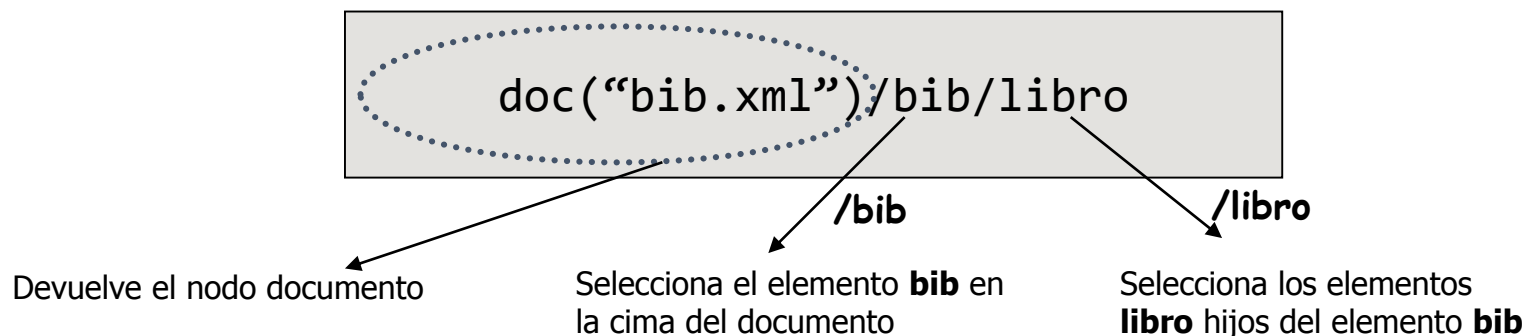
(:Devuelve el documento completo, el nodo documento :)

(:El documento se identifica por su URI, *Universal Resource Identifier*:)

- **fn:collection(URI)**. 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")
```

- Expresiones constituidas por **Path Expressions** para localizar nodos:
 - 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”



- Un **predicado** es una expresión encerrada entre corchetes que se emplea para **filtrar una secuencia de valores**
 - Ej1: **libro[titulo="mi titulo"]** -> Selecciona los libros cuyo título sea "mi título"
 - Ej2: **libro[precio>10]** -> Selecciona los libros cuyo precio sea superior a 10
 - Ej3: **libro[5]** -> Selecciona el quinto nodo hijo
 - Ej4: **libro[precio]** -> Selecciona los libros que tienen un nodo hijo precio
 - Ej5: **libro[@año>1997]** -> Selecciona los libros cuyo año es posterior al 97

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]
```

1. Obtener el primer autor que aparece en el documento

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

2. Obtener los años en los que fueron publicados los libros que aparecen en bib.xml

```
doc("bib.xml")/bib/libro/@año
```

3. Obtener los libros cuyos autores se apellidan Korth

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

4. Mostrar la cantidad total de libros que hay en bib.xml


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

Creación de nodos

- XQuery permite crear nuevos nodos
- Existen **dos modos de creación** de elementos:
 - Escribiendo las etiquetas de comienzo y cierre
 - Utilizando los constructores de elementos y atributos
- Cuando se utiliza un constructor las expresiones que se utilizan para componerlos y que tienen que ser evaluadas deben estar entre { } para que no sean tomadas como literales


```
<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> {string(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>
```

1. 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>
```

Empleando constructores de elementos y atributos

```
element titulos {  
  attribute num { count  
    (doc("bib.xml")//titulo) },  
  element titulo  
    {doc("bib.xml")//titulo}  
}
```

Es equivalente a:

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

Expresiones FLWOR

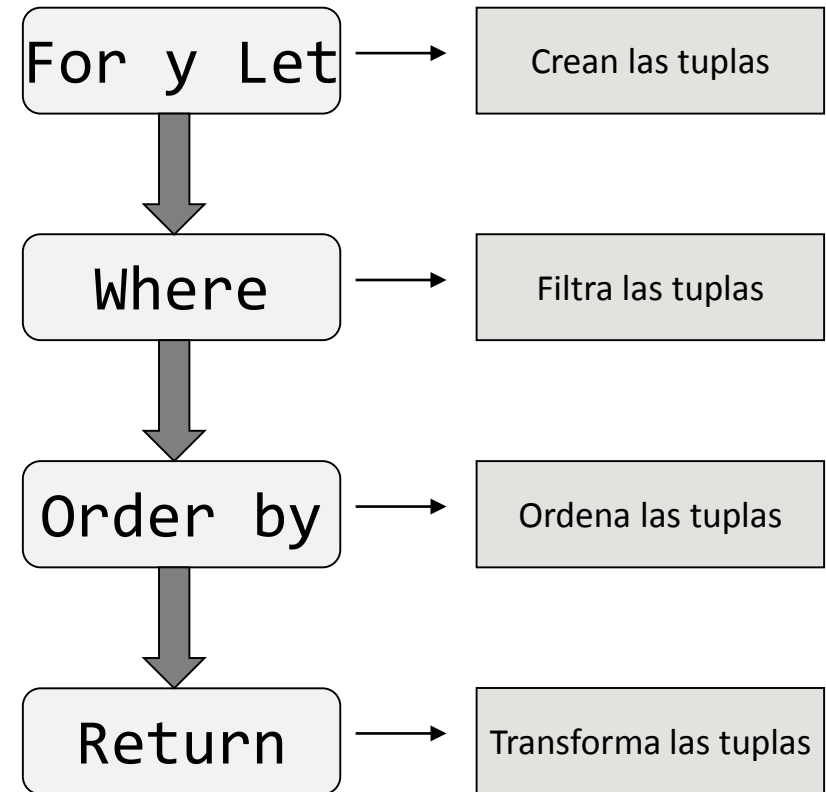
- Permiten la combinación y re-estructuración de los nodos
- FLWOR (***F**or **L**et **W**here **O**rders **R**eturn*) es una de las expresiones más potentes y típicas de XQuery
- Se basa en ligar valores a variables con las **cláusulas for y let** y emplear esas variables para crear nuevos resultados

1. 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
```

Expresiones FLWOR (II)

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



- El objetivo de esta cláusula es obtener una secuencia de tuplas (nodos del documento origen)
- Sintaxis:
 - for** **'var'** **in** **'binding-sequence'** siendo,
 - **var**: variable sobre la que se almacenan los nodos
 - **binding-sequence** los nodos que se van a tratar
- Se pueden especificar varias variables en la misma sentencia, con lo que el resultado es el producto cartesiano de todas ellas, es decir, las tuplas generadas cubren todas las posibles combinaciones de los nodos de dichas variables

Cláusula **for** - Ejemplos

- Ejemplo 1:

```
for $a in doc("bib.xml")//titulo  
  return <best-sellers> {$a} </best-sellers>
```

- Ejemplo 2:

```
for $a in doc("bib.xml")//titulo, $b in  
doc("bib.xml")//precio  
  return <best-sellers> {$a, $b} </best-sellers>
```

- Importante: a) devuelve todas las combinaciones posibles de las dos secuencias y b) la sentencia **return** se evalúa para cada una de las tuplas

Cláusula **for** – Variable posicional **at**

- La cláusula **for** soporta variables posicionales que identifican la posición de un elemento en la secuencia de tuplas generada
- Se especifica con la partícula **at**

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


- El objetivo de esta clausula, al igual que la **for**, es obtener una secuencia de tuplas (nodos del documento origen) pero la forma de hacerlo es distinto
- Liga variables al resultado entero de una expresión y devuelve una única tupla
- Sintaxis:

let \$var := 'binding-sequence'

- Mientras que con la cláusula **for** obtenemos una tupla con cada elemento de entrada, con la cláusula **let** obtenemos una única tupla

- Ejemplo 1:

```
let $c := doc ("bib.xml")//titulo  
return <best-sellers> {$c} </best-sellers>
```

- Ejemplo 2:

```
let $b := doc ("bib.xml")//titulo, $c:= doc("bib.xml")//precio  
return <best-sellers> {$b,$c} </best-sellers>
```

- La sentencia return solo se evalúa para una única tupla y por tanto solamente se obtiene un elemento best-sellers

- Las sentencias **for** y **let** se pueden emplear conjuntamente
- En este caso para cada tupla devuelta por la sentencia **for** se evalúa la sentencia **let**, y después se evalúa **return**
- Ejemplo

```
for $a in doc("bib.xml")//titulo
  let $c := $a/../../autor (: ó bien $a/ancestor::libro/autor :)
  return <best-sellers> {$a,$c} </best-sellers>
```

Cláusula where

- Es opcional
- 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
- Sintaxis:

where 'expresion'

- Ejemplo:

```
for $a in doc("bib.xml")//libro where $a/precio<70  
  return <best-sellers> {$a/titulo} </best-sellers>
```

- Es opcional
- Es ejecutada una vez para cada tupla retenida por la cláusula **where** y es evaluada antes de la cláusula **return**
- La forma de ordenar la secuencia de tuplas viene dada a través de las *order-spec*:
 - **order by** 'expr' 'order-modifier'
 - **stable order by** 'expr' 'order-modifier'
- Si se especifica *stable* indica que en caso de existir dos tuplas que contengan valores idénticos en todos los *order-spec* que hemos empleado se mantenga el orden de entrada

Cláusula **order by** (II)

- El orden de la ordenación puede ser:
 - **ascending**
 - **descending**
- Se puede especificar como tratar las tuplas que no contengan el elemento por el que se ordena:
 - **empty greatest**
 - **empty least**
- Ejemplo:

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

1. 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
```

2. 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
```

Cláusula return

- Esta cláusula es **ejecutada una vez** para cada tupla dentro de la secuencia de tuplas obtenida con las sentencias **for**, **let**, **where** y en el orden especificado por **order by**
- 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 $b in doc("bib.xml")//libro/autor
  return <autor> {string($b/nombre), " ", string($b/apellido)} </autor>
```


1. Listar el título y precio de los libros cuyos precios sean inferiores a 100 euros.

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

2. 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>
```

3. 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>
```

4. 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>
```

5. 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>
```

6. Obtener el título, el año y la editorial 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>
```

7. Obtener los títulos de los libros y precio para aquellos que tengan unos precios inferiores o iguales 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>
```

Expresiones condicionales

- XQuery soporta expresiones condicionales del tipo

```
if (expresion) then (se ejecuta si la expresión devuelve true)
else (se ejecuta si la expresión devuelve false o una secuencia vacía)
```

- La cláusula ***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 ()
```

Ejemplo de expresiones condicionales

1. 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>
```

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 evalúa 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

1. Listar los títulos de los libros en los que al menos uno de sus autores se apellide Silberschatz

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

2. Listar los títulos de los libros en los que todos sus autores se apelliden Silberschatz

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


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. \$nodo1<<\$nodo2 es true si el nodo ligado a \$nodo1 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. idiv es para divisiones con enteros en la que se ignora el resto.
- Negación: **not**
 - Es una función más que un operador. Invierte un valor booleano.

- **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

1. 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.

Ejemplos de operadores (II)

- **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.

2. Muestra todos los datos del libro “Fundamentos de Bases de Datos” excepto sus autores

```
for $b in doc("bib.xml")//libro
where $b/titulo="Fundamentos de Bases de Datos"
return
  <libro_resumen>
    {$b/* except $b/autor }
  </libro_resumen>
```

- **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()`, `exists()`, ...
- Etc.

1. Listar todos los apellidos distintos de los autores

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

2. Listar todos los títulos de los libros que tengan al menos un autor

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

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

3. Mostrar los títulos de los libros que no tiene editor

```
<sin-editor>
{for $a in doc("bib.xml")//libro
  where empty ($a/editor)
  return $a/titulo
}
</sin-editor>
```

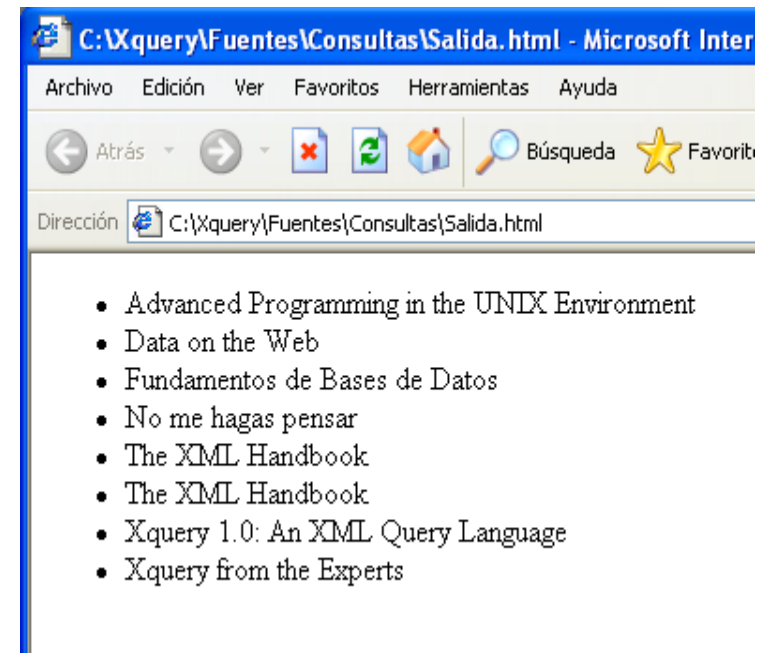
4. Mostrar los libros cuyos precios superan la media

```
let $l := doc("bib.xml")//libro
let $c:= avg($l//precio)
return <libros-superan-media> {$l[precio>$c]} </libros-superan-media>
```

Ejemplo generación HTML (I)

5. 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>
```



6. 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 border="1">{
    for $b in doc("bib.xml")//libro
      return
        <tr> <td> <i> {string($b/titulo)} </i> </td> </tr>}
  </table>
</body>
</html>
```


- Base de datos open-source nativa XML creada por Wolfgang Meier (2000)
- <http://exist-db.org>
- Proporciona un motor propio y optimizado para **Xquery**
- Puede ser desplegada como un servidor de base de datos **stand-alone**, como una librería **Java embebida** o como **parte de una aplicación web** (ejecutándose en el motor de *servlets*)
- **eXide** es el XQuery IDE
- eXist permite el desarrollo y distribución de aplicaciones web



- Descargar la base de datos
 - <http://exist-db.org/exist/apps/homepage/index.html>
- Instalación
 - <http://exist-db.org/exist/apps/doc/quickstart.xml>
- Sobre el dashboard
 - <http://exist-db.org/exist/apps/doc/dashboard.xml>
- Guía rápida para construcción de aplicaciones con eXide
 - <http://exist-db.org/exist/apps/doc/development-starter.xml>
- Como funciona el HTML Templating Module
 - <http://exist-db.org/exist/apps/doc/templating.xml>
- XUpdate
 - http://exist-db.org/exist/apps/doc/update_ext.xml

- XQuery 1.0: An XML Query Language
 - <http://www.w3.org/TR/xquery/>
- Xquery
 - Walmsley Priscilla. O`Reilly Media Inc. Marzo 2007
- XQuery. The XML Query Language
 - Brundage Michael Chamberlin .Addison-Wesley, Professional. Febrero 2004
- Xquery from the Experts. A Guide to the W3C XML Query Language.
 - Chamberlin D., Draper D., et. Al. Addison-Wesley, 2005.

