

BASES DE DATOS NATIVAS.

- **Bases de datos nativas**
- **Base de datos Exist**
- **XPath**
 - Rutas absolutas
 - Rutas relativas
 - Operador *
 - Condiciones de selección
 - Sobre los nodos
 - Funciones y expresiones matemáticas
- **XQuery**
 - Expresiones flwor
 - Clausulas for, let, where, order by, return
 - Expresiones condicionales
 - Algunos operadores
 - Xquery resultados en HTML
 - Funciones
 - Consultas complejas con Xquery
 - Joins de documentos
 - Uso de varios for
 - Sentencias de actualización de eXists
 - o Insert
 - o Replace
 - o Value
 - o Delete
 - o Rename
- **API XQJ** (Acceso a BD nativas desde java)

Bases de datos nativas

A diferencia de las bases de datos relacionales (centradas en los datos), las bases de datos nativas XML, no poseen campos ni almacenan datos, lo que almacenan son documentos XML, son bases de datos centradas en documentos y la unidad mínima de almacenamiento es el documento XML.

Una base de datos nativa cumple lo siguiente:

- Define un modelo lógico para un documento XML.
- Tiene una relación transparente con el mecanismo de almacenamiento, que debe incorporar las características ACID de cualquier SGBD (Atomicity, Consistency, Isolation an Durability: Atomicidad, consistencia, aislamiento y durabilidad)
- Permite tecnologías de consulta y transformación propias de XML XQuery, XPath, XSLT.

Ventajas de las bases de datos XML:

- Ofrecen un acceso y almacenamiento de información ya en formato XML, sin necesidad de incorporar código adicional.
- La mayoría incorpora un motor de búsqueda de alto rendimiento.
- Es muy sencillo añadir documentos XML al repositorio.
- Se pueden almacenar datos heterogéneos.

Desventajas de las bases de datos XML:

- Puede resultar difícil indexar documentos para realizar búsquedas.
- No suelen ofrecer funciones para la agregación.
- Se suele almacenar la información como un documento o como un conjunto de nodos, por lo que su síntesis para formar nuevas estructuras sobre la marcha puede resultar complicada y lenta.

En la actualidad la mayoría de los SGBD incorporan mecanismos para extraer y almacenar datos en formato XML.

Ejemplos de soporte XML en Oracle y MySql

Consulta devuelve las filas de tabla EMP en formato XML

Select

```
XMLELEMENT("EMP_ROW",XMLFOREST(EMPNO,ENAME,JOB,MGR,HIREDATE,SAL,COM  
M,DEPTNO)) FILA_EN_XML FROM EMP;
```

Creación de una tabla almacenada del tipo xml (el tipo de dato xmltype permite almacenar y consultar datos xml)

```
CREATE TABLE TABLA_XML_PRUEBA(COD NUMBER, DATOS XMLTYPE);
```

Insertar filas en formato XML

```
INSERT INTO TABLA_XML_PRUEBA  
VALUES(1,XMLTYPE('<FILA_EMP><EMP_NO>123</EMP_NO><APELLIDO>RAMOS  
MARTIN</APELLIDO><OFICIO>PROFESORA</OFICIO><SALARIO>1500</SALARIO></  
FILA_EMP>'));
```

```
INSERT INTO TABLA_XML_PRUEBA  
VALUES(1,XMLTYPE('<FILA_EMP><EMP_NO>124</EMP_NO><APELLIDO>GARCIA  
SALADO</APELLIDO><OFICIO>FONTANERO</OFICIO><SALARIO>1700</SALARIO></  
FILA_EMP>'));
```

Para extraer datos de la tabla se usan expresiones XPATH:

Visualizar los apellidos de los empleados

```
SELECT EXTRACTVALUE(DATOS,'/FILA_EMP/APELLIDO') FROM TABLA_XML_PRUEBA;
```

Visualizar el nombre del empleado con numero de empleado 123

```
SELECT EXTRACTVALUE(DATOS,'/FILA_EMP/APELLIDO') FROM TABLA_XML_PRUEBA  
WHERE EXISTSNODE(DATOS,'/FILA_EMP[EMP_NO=123]')=1;
```

Base de datos eXist

eXist es un SGBD libre de código abierto que almacena datos XML de acuerdo a un modelo de datos XML.

El motor de la base de datos está completamente escrito en java, soporta los estándares de consulta XPath, XQuery y XSLT.

Los documentos XML se almacenan en colecciones, las cuales pueden estar anidadas, desde un punto de vista práctico el almacén de datos funciona como un sistema de ficheros. Cada documento está en una colección. No es necesario que los documentos tengan una DTD o un XML Schema asociado y dentro de una colección puede haber documentos de cualquier tipo.

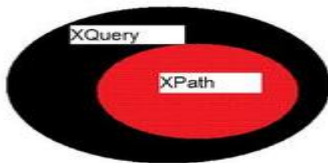
En la carpeta eXist\webapp\WEB-INF\data es donde se guardan los ficheros más importantes de la BD:

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

Instalación de eXist ver documento adjunto.

XPath:

XPath es un lenguaje que permite seleccionar nodos de un documento XML y calcular valores a partir de su contenido. Existen dos versiones de XPath aprobadas por el W3C.



La forma en que XPath selecciona partes del documento XML se basa en la representación arbórea que se genera del documento.

A la hora de recorrer un árbol XML, podemos encontrarnos con los siguientes tipos de nodos:

- **nodo raíz o "/"**: primer nodo del documento xml.
- **nodo elemento**: cualquier elemento de un documento xml, es un nodo elemento en el árbol, son las etiquetas del árbol.
- **nodo texto**: Los caracteres que están entre las etiquetas.
- **nodo atributo**: Son propiedades añadidas a los nodos elemento, se representan con @
- **nodo comentario**: las etiquetas de comentario
<!--comentario-->
- **Nodo espacio de nombres**, contienen espacios de nombres.
- **Nodo instrucción de proceso**, contiene instrucciones de proceso, va entre las etiquetas <?.....?>

Ejemplo

```
<?xml version="1.0" encoding="ISO-8859-1"?> ← nodo instrucción de proceso
<universidad> ← nodo raíz
  <departamento telefono="112233" tipo="A"> ← nodo atributo
    <codigo>IFC1</codigo>
    <nombre>Informática</nombre> ← nodo elemento
    <empleado salario="2000">
      <puesto>Asociado</puesto>
      <nombre>Juan Parra</nombre> ← nodo texto
    </empleado>
    <empleado salario="2300">
      <puesto>Profesor</puesto>
      <nombre>Alicia Martín</nombre>
    </empleado>
  </departamento>
```

Rutas absolutas comienzan desde la raíz se indican con /

Rutas relativas pueden empezar en cualquier parte de la colección //, por ejemplo //dept_no devolverá los elementos dept_no tanto del fichero departamentos.xml como empleados.xml

Operador * se usa para nombrar cualquier etiqueta, se usa de comodín.

Condiciones de selección: se usan corchetes para seleccionar elementos concretos, los comparadores serán <, >, <=, >=, =, !=, or, and y not(escritos en minúsculas)

El separador | se usa para unir varias rutas.

Sobre los nodos:

Text() selecciona el contenido del elemento

Node() selecciona todos los nodos, los elementos y el texto

Prefix:* para seleccionar nodos con un espacio de nombres determinado

Processing-instruction() selecciona nodos que son instrucciones de proceso

Comment() selecciona nodos de tipo comentario

Funciones y expresiones matemáticas

last() selecciona el último elemento del conjunto seleccionado.

position() devuelve un número igual a la posición del elemento actual

count() cuenta el número de elementos seleccionados

sum() devuelve la suma del elemento seleccionado

max(), min(), avg() calculan el máximo, mínimo y media respectivamente del elemento seleccionado

name() devuelve el nombre del elemento seleccionado

concat(cad1,cad2) concatena las cadenas

starts-with(cad1,cad2) es verdadera cuando la cadena cad1 tiene como prefijo a la cad2

contains(cad1,cad2) es verdadera cuando la cadena cad1 contiene a la cad2

string-length(argumento) devuelve el número de caracteres de su argumento

div() realiza divisiones en punto flotante

mod() calcula el resto de la división

data(expresión XPath) devuelve el texto de los nodos de la expresión sin etiquetas, también se usa para extraer el contenido de los atributos.

number(argumento) convierte a número el argumento que puede ser cadena, booleano o un nodo.

abs(num) devuelve el valor absoluto de un número

ceiling(num) devuelve el entero más pequeño mayor o igual que la expresión numérica especificada

floor(num) devuelve el entero más grande que sea menor o igual que la expresión numérica especificada

round(num) redondea el valor de la expresión numérica

string(argumento) convierte el argumento en cadena

compare(expr1,expr2) compara las dos expresiones, devuelve 0 si son iguales, 1 si $\text{exp1} > \text{exp2}$ y -1 $\text{exp1} < \text{exp2}$

substring(cadena,comienzo,num) extrae la cadena desde la posición indicada en comienzo tantos caracteres como le indique num

lower-case(cadena) convierte a minúsculas la cadena

upper-case(cadena) convierte a mayúsculas la cadena

translate(cadena1,caract1,caract2) reemplaza dentro de cadena1, los caracteres que se expresan en caract1, por los correspondientes que aparecen en caract2, uno por uno.

Ends-with(cadena1,cadna2) devuelve true si la cadena1 termina en cadena2

year-from-date(fecha) devuelve el año de la fecha AAAAMMDD

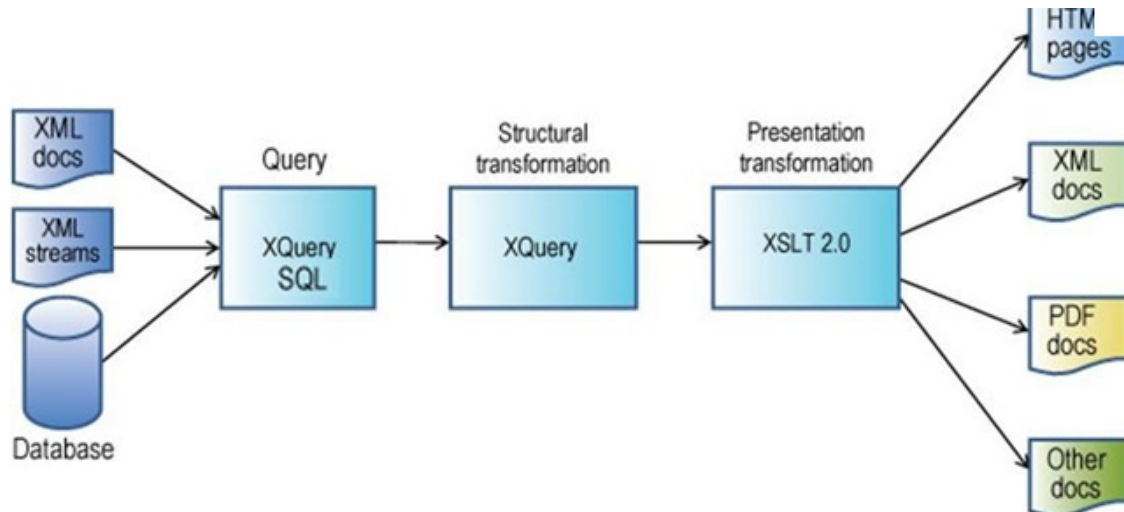
month-from-date(fecha) devuelve el mes de la fecha

day-from-date(feh) devuelve el día de la fecha

Xquery:

Una consulta XQuery es una expresión que lee datos de uno o más documentos en XML y devuelve como resultado otra secuencia de datos en XML.

XQuery contiene a XPath, toda expresión de consulta en XPath es válida y devuelve el mismo resultado en XQuery.



- Xquery es sensible a mayúsculas y minúsculas
- cualquier elemento, atributo o variable de xquery tiene que ser un identificador válido en xml
- un valor string o cadena de Xquery puede estar contenido tanto en comillas simples como dobles
- una variable Xquery se define con un símbolo \$ seguido por un nombre. por ejemplo, \$bib
- un comentario Xquery se inserta dentro de un limitador de inicio (: y un delimitador de fin :). por ejemplo, (: hola :). el procesador no interpreta estos comentarios.

EXPRESIONES FLWOR

En XQuery las consultas siguen al norma FLWOR, corresponde a las siglas de For, Let, Where, Order y Return. Permite a diferencia de XPath manipular, transformar y organizar los resultados de las consultas.

For	Vincula una o más variables a expresiones escritas en XPath, creando un flujo de tuplas en el que cada tupla está vinculada a una de las variable.
Let	Vincula una variable al resultado completo de una expresión añadiendo esos vínculos a las tuplas generadas por una cláusula for o, si no existe ninguna cláusula for, creando una única tupla que contenga esos vínculos.
Where	Filtra las tuplas eliminando todos los valores que no cumplan las condiciones dadas.
Order by	Ordena las tuplas según el criterio dado.
Return	Construye el resultado de la consulta para una tupla dada, después de haber sido filtrada por la cláusula where y ordenada por la cláusula order by.

LA CLÁUSULA FOR

- Se usa para seleccionar nodos y almacenarlos en una variable, es similar a la cláusula from de SQL. Dentro de for se escribe una expresión Xpath que selecciona los nodos.

XQUERY	XPATH
For \$emp in /EMPLEADOS/EMP_ROW Return \$emp (devuelve los elementos EMP_ROW)	/EMPLEADOS/EMP_ROW (devuelve los elementos EMP_ROW)
For \$emp in /EMPLEADOS/EMP_ROW Return \$emp/APELLIDO (devuelve los apellidos de los empleados)	/EMPLEADOS/EMP_ROW /APELLIDO (devuelve los apellidos de los empleados)

- 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.
- la cláusula for enlaza una variable con cada item que se devuelve por la expresión in. esta cláusula produce una iteración. la palabra for indica que se va a realizar una iteración en un árbol xml. la palabra clave **in** indica sobre qué árbol se va a ejercer dicha iteración, con la instrucción return se recupera un resultado.

**for \$x in /bib/libro/titulo
where \$x/precio>30
return \$x/titulo**

Devuelve el titulo de los libros de precio mayor 30

- puede haber múltiples cláusulas for en una misma expresión
- para conseguir una iteración un determinado número de veces se usa la palabra reservada **to**. por ejemplo:

```
for $x in (1 to 3)
return <test>{$x}</test>
```

- la palabra reservada **at** permite contar la iteración:

```
for $x at $i in /bib/libro/titulo
return <libro>{$i}. {data($x)}</libro>
```

DEVUELVE EL RESULTADO:

```
<libro>1. La vuelta al mundo en 80 días</ libro >
<libro>2. HARRY POTTER</ libro >
< libro >3. Los cinco</libro >
< libro >4. Los miserables</libro >
```

- se permite también más de una expresión en una cláusula for. para ello, se separan los valores mediante comas, entonces se realizará el producto cartesiano:

```
for $x in (10,20), $y in (100,200)
return <test>x={$x} and y={$y}</test>
```

LA CLÁUSULA LET

- Permite que se asignen valores resultantes de expresiones XPath a variables para simplificar la representación. Se puede poner varias líneas let una por cada variable o separar las variables por comas.
- Liga variables al resultado entero de una expresión y devuelve una única tupla.
- la cláusula let permite hacer asignaciones de variables, permitiendo evitar el tener que repetir la misma expresión varias veces
- la cláusula let no provoca una iteración
- Si una cláusula let aparece en una consulta que ya posee una o más cláusulas for, los valores de la variable vinculada por la cláusula let se añaden a cada una de las tuplas generadas por la cláusula for

Diferencias entre las cláusulas for y let.

La cláusula **for** vincula una variable con cada nodo que encuentre en la colección de datos.

En este ejemplo la variable \$d va vinculándose a cada uno de los títulos de todos los libros del archivo "libros.xml", creando una tupla por cada título. Por este

motivo aparece repetido el par de etiquetas <titulos>...</titulos> para cada título.

let, en cambio, vincula una variable con todo el resultado de una expresión. En este ejemplo, la variable \$d se vincula a todos los títulos de todos los libros del archivo

"libros.xml", creando una única tupla con todos esos títulos. Por este motivo solo aparece el par de etiquetas <titulos>...</titulos> una única vez.

Con for

```
for $d in /bib/libro/titulo
return
<titulos>{ $d }</titulos>
```

Send Clear Check

Found 4 in 0 seconds.

```
<<
1 <titulos>
  <titulo>TCP/IP Illustrated</titulo>
</titulos>
2 <titulos>
  <titulo>Advan Programming for Unix environment</titulo>
</titulos>
3 <titulos>
  <titulo>Data on the Web</titulo>
</titulos>
4 <titulos>
  <titulo> Economics of Technology for Digital TV</titulo>
</titulos>
```

Con let

```
let $d := /bib/libro/titulo
return
<titulos>{ $d }</titulos>
```

Send Clear Check

Found 1 in 0 seconds.

```
<<
1 <titulos>
  <titulo>TCP/IP Illustrated</titulo>
  <titulo>Advan Programming for Unix environment</titulo>
  <titulo>Data on the Web</titulo>
  <titulo> Economics of Technology for Digital TV</titulo>
</titulos>
```

LA CLÁUSULA WHERE

- filtra las tuplas producidas por las cláusulas let y for.
- contiene una expresión que es evaluada para cada tupla. si su evaluación es false esa tupla es descartada
- la cláusula where se usa para especificar uno o más criterios para el resultado **where \$x/price>30 and \$x/price<100**

LA CLÁUSULA ORDER BY

- La cláusula order by especifica el criterio de ordenación del resultado. Así, para ordenar ejemplo por categoría y título:
- 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 $x in /bib/libro
order by $x/title
return $x/title
```

LA CLÁUSULA RETURN

- La cláusula return indica qué es lo que se quiere devolver.
- 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.
- Si añadimos etiquetas los datos a visualizar van entre llaves{}
- En la cláusula return se pueden añadir condicionales usando if-then-else

EXPRESIONES CONDICIONALES

"if-then-else"

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

- La cláusula else es obligatoria y debe aparecer siempre en la expresión condicional. Si no hay instrucciones para añadirle se puede indicar como **else()**. Esto se debe a que toda expresión Xquery debe devolver un valor.

Ejemplo:

```
For $dep in /universidad/departamento
Return if ($dep/@tipo='A')
Then <tipoA>{data($dep/nombre)}</tipoA>
else()
```

Devuelve los departamentos de tipo A entre etiquetas

- las expresiones condicionales “**if-then-else**” se permiten limitar los resultados a las condiciones de las expresiones if-then-else en xquery.

```
for $x in bib/libro
return if ($x/@categoria="NIÑOS")
then <NIÑO>{data($x/title)}</NIÑO>
else <ADULTO>{data($x/title)}</ADULTO>
```

ALGUNOS OPERADORES

- Comparación de valores:** **EQ** (equal), **NE**(no equal o distinto de), **LT**,(less than o inferior a), **LE**(inferior o igual), **GT**(greater than o superior a), **GE** (superior o igual) . 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 ordenes 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 \$node2

- Lógicos: and y or**

Se emplean para combinar condiciones lógicas dentro de un predicado. Ej. Item[seller="Smith" and precio]

- Aritméticos: +, -, *, div, mod**

Son definidos sobre valores numéricos

- Negación: not**

Es una función más que un operador. Invierte un valor booleano. _No es lo mismo comparar con un método que con otro, puesto que cada una tiene una indicación específica

La diferencia entre los dos métodos de comparación se muestra a continuación

```
$bib//libro/@q > 10
```

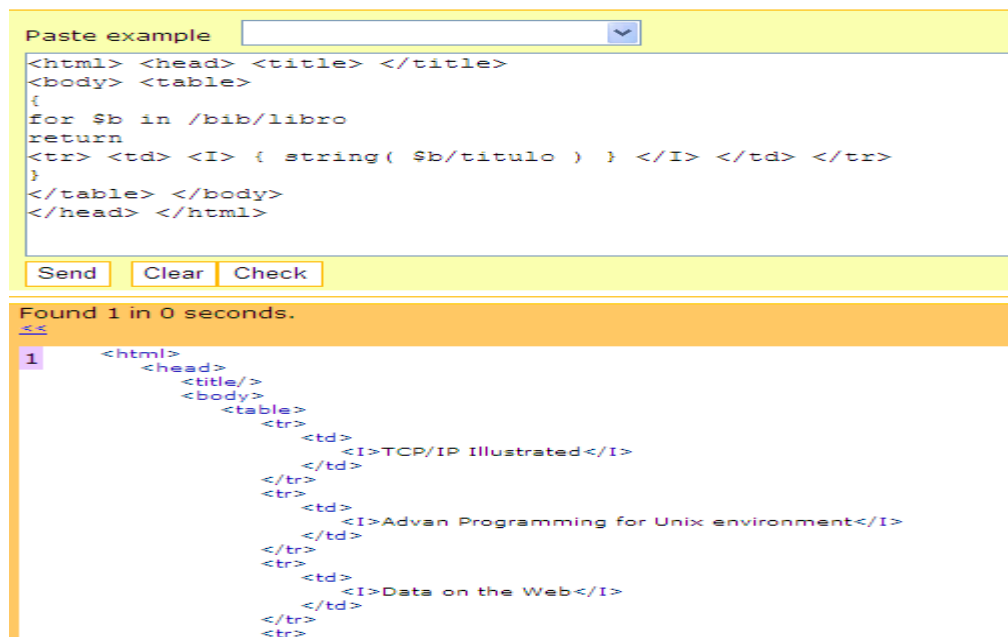
```
$bib//libro/@q gt 10 (para salidas de un solo valor)
```

La primera expresión devuelve true si cualquier atributo q tiene un valor mayor que 10

La segunda expresión devuelve true si hay un solo atributo q devuelto por la expresión y su valor es mayor que 10. Si hay más de un atributo q devuelto, se produce un error .

XQUERY resultados en formato HTML

- para crear consultas xquery que después de ejecutarlas nos devuelva los resultados en formato html.
- se pueden unir dentro de xquery etiquetas html, y expresiones flwor.
- cuando se fusionan en una consulta, se le indica al motor de consultas que parte es la que tiene que procesar como consulta.
- se indica con llaves "{ }", que parte es flwor.



```
Paste example 
<html> <head> <title> </title>
<body> <table>
{
for $b in /bib/libro
return
<tr> <td> <I> { string( $b/titulo ) } </I> </td> </tr>
}
</table> </body>
</head> </html>

Send Clear Check

Found 1 in 0 seconds.
1
<html>
  <head>
    <title/>
  <body>
    <table>
      <tr>
        <td>
          <I>TCP/IP Illustrated</I>
        </td>
      </tr>
      <tr>
        <td>
          <I>Advan Programming for Unix environment</I>
        </td>
      </tr>
      <tr>
        <td>
          <I>Data on the Web</I>
        </td>
      </tr>
    </table>
  </body>
</html>
```

FUNCIONES EN XQUERY

Xquery incluye más de 100 funciones propias, se pueden ver en:

<http://www.w3.org/2005/02/xpath-functions>

El prefijo usado por defecto en el espacio de nombre de las funciones es **fn:**

No es necesario emplear el prefijo cuando son llamadas.

- Una llamada a una función puede aparecer en el mismo sitio donde una expresión pueda colocarse. Por ejemplo:

- En un elemento

```
<name>{uppercase($booktitle)}</name>
```

- En el predicado de una expresión path

```
doc("books.xml")/bookstore/book[substring(title,1,5)='Harry']
```

- En una cláusula let

```
let $name := (substring($booktitle,1,4))
```

Matemáticos:	+, -, *, div(*), idiv(*), mod.
Comparación:	=, !=, <, >, <=, >=, not()
Secuencia:	union (), intersect, except
Redondeo:	floor(), ceiling(), round().
Funciones de agrupación:	count(), min(), max(), avg(), sum().
Funciones de cadena:	concat(), string-length(), starts-with(), ends-with(), substring(), upper-case(), lower-case(), string()
Uso general:	distinct-values(), empty(), exists()

Principales operadores y funciones de XQuery

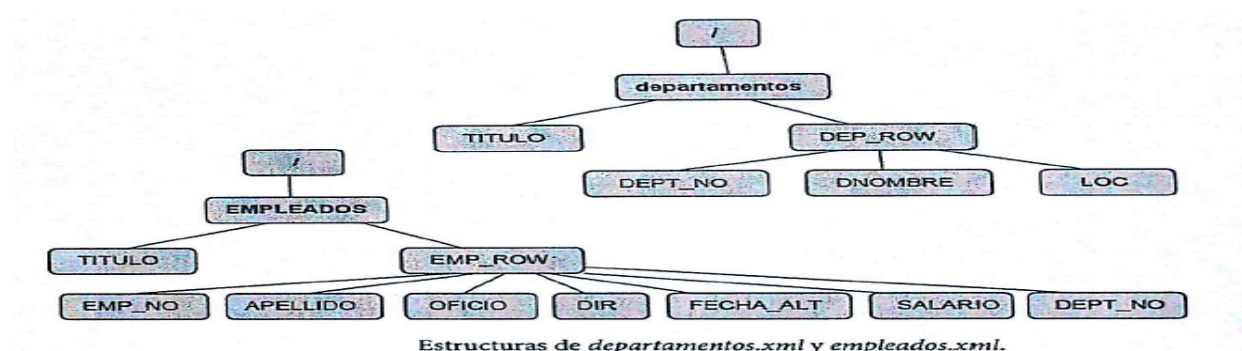
- XQuery soporta dos cuantificadores existenciales llamados "some" y "every", de tal manera que nos permite definir consultas que devuelva algún elemento que satisfaga la condición ("**some**") o consultas que devuelvan los elementos en los que todos sus nodos satisfagan la condición ("**every**").
- El operador de sustracción (**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.
- La función **distinct-values()** extrae los valores de una secuencia de nodos y crea una nueva secuencia con valores únicos, eliminando los nodos duplicados.
- La función **empty()** devuelve cierto cuando la expresión entre paréntesis está vacía
- La función opuesta a empty() es **exists()**, la cual devuelve cierto cuando una secuencia contiene, al menos, un elemento.

CONSULTAS COMPLEJAS EN XQUERY

Dentro de las consultas Xquery podremos trabajar con varios documentos XML para extraer su información, podremos incluir tantas sentencias for como se deseen incluso dentro del return. Además podemos añadir, borrar e incluso modificar elementos, bien generando un documento xml nuevo o utilizando las sentencias de actualización de eXists.

Ejemplos de Joins de documentos.

- Visualizar por cada empleado del documento **empleados.xml**, su apellido, su número de departamento y nombre del departamento que se encuentra en el documento **departamentos.xml**
- Obtener por cada departamento, el nombre del departamento, el número de empleados y la media del salario
- Obtener por cada departamento el nombre del empleado que más gana.



Soluciones:

a)

```
for $emp in (/EMPLEADOS/EMP_ROW)
let $emple:=$emp/APELLIDO
let $dep:=$emp/DEPT_NO
let $ndep:=(/departamentos/DEP_ROW[DEPT_NO=$dep]/DNOMBRE)
return <res>{$emple,$dep,$ndep}</res>
```

```
<res>
  <APELLIDO>SANCHEZ</APELLIDO>
  <DEPT_NO>20</DEPT_NO>
  <DNOMBRE>INVESTIGACION</DNOMBRE>
</res>

<res>
  <APELLIDO>ARROYO</APELLIDO>
  <DEPT_NO>30</DEPT_NO>
  <DNOMBRE>VENTAS</DNOMBRE>
</res>

<res>
  <APELLIDO>SALA</APELLIDO>
  <DEPT_NO>30</DEPT_NO>
  <DNOMBRE>VENTAS</DNOMBRE>
</res>

</res>
```

b)

```
for $dep in /departamentos/DEP_ROW
let $d:=$dep/DEPT_NO
let $tot:=sum(/EMPLEADOS/EMP_ROW[DEPT_NO=$d]/SALARIO)
let $cur:=count(/EMPLEADOS/EMP_ROW[DEPT_NO=$d]/EMP_NO)
return
  <result>{$dep/DNOMBRE}
<sumsalario>{$tot}</sumsalario>
<totemple>{$cur}</totemple>
</result>
```

```
<result>
  <DNOMBRE>CONTABILIDAD</DNOMBRE>
  <sumsalario>8675</sumsalario>
  <totemple>3</totemple>
</result>

<result>
  <DNOMBRE>INVESTIGACION</DNOMBRE>
  <sumsalario>11370</sumsalario>
  <totemple>5</totemple>
</result>

<result>
  <DNOMBRE>VENTAS</DNOMBRE>
  <sumsalario>10415</sumsalario>
  <totemple>6</totemple>
</result>
```

c)

```
for $emp in /EMPLEADOS/EMP_ROW
let $d:=$emp/DEPT_NO, $nom:=$emp/APELLIDO, $sal:=$emp/SALARIO
let $ndep:=(/departamentos/DEP_ROW[DEPT_NO=$d]/DNOMBRE)
let $salmax:=max(/EMPLEADOS/EMP_ROW[DEPT_NO=$d]/SALARIO)
return
if($sal=$salmax) then
<depart>
{data($ndep)}
<salmax>{data($sal)}</salmax>
  <emple>{data($nom)}</emple>
</depart>
else()
```

```
<depart>VENTAS
  <salmax>3005</salmax>
  <emple>NEGRO</emple>
</depart>
```

```
<depart>INVESTIGACION
  <salmax>3000</salmax>
  <emple>GIL</emple>
</depart>
```

```
<depart>CONTABILIDAD
  <salmax>4100</salmax>
  <emple>REY</emple>
</depart>
```

```
<depart>INVESTIGACION
  <salmax>3000</salmax>
  <emple>FERNANDEZ</emple>
</depart>
```

Ejemplos de uso de varios for:

La utilización de varios for es muy útil para consultas en documentos XML anidados y también cuando utilizamos varios documentos unidos por una clausula where como una combinación de tablas en SQL.



Árbol del documento *universidad.xml*.

- a) Visualizar por cada departamento del documento *universidad.xml*, **el número de empleados que hay en cada puesto de trabajo.**

Se usa un for para obtener los nodos departamento y un segundo for para obtener los distintos puestos de cada departamento.

```
for $dep in /universidad/departamento
for $pue in distinct-values($dep/empleado/puesto)
let $cur:=count($dep/empleado[puesto=$pue])
return <depart>{data($dep/nombre)}
    <puesto>{data($pue)}</puesto>
    <profes>{$cur}</profes>
</depart>
```

```
<depart>Informática
  <puesto>Asociado</puesto>
  <profes>1</profes>
</depart>
<depart>Informática
  <puesto>Profesor</puesto>
  <profes>1</profes>
</depart>
<depart>Matemáticas
  <puesto>Técnico</puesto>
  <profes>1</profes>
</depart>
<depart>Matemáticas
  <puesto>Profesor</puesto>
  <profes>2</profes>
</depart>
<depart>Matemáticas
  <puesto>Tutor</puesto>
  <profes>1</profes>
</depart>
```

- b) Visualizar por cada departamento del documento universidad.xml, **el salario máximo y el empleado que tiene ese salario**. El primer for obtiene los nodos departamento y el segundo for los empleados de cada departamento. Para sacar el máximo, en la salida preguntamos si el salario es el máximo.

```
for $dep in /universidad/departamento
for $emp in $dep/empleado
let $emple := $emp/nombre
let $sal:=$emp/@salario
return if ($sal=$dep/max(empleado/@salario))
then
  <depart>{data($dep/nombre)}<salamax>{data($sal)}</salamax>
    <empleado>{data($emple)}</empleado></depart>
else()
```

```
<depart>Informática
  <salamax>2300</salamax>
  <empleado>Alicia Martín</empleado>
</depart>
<depart>Matemáticas
  <salamax>2500</salamax>
  <empleado>Antonia González</empleado>
</depart>
<depart>Análisis
  <salamax>2200</salamax>
  <empleado>Mario García</empleado>
</depart>
```

- c) Visualiza por cada puesto del documento universidad.xml, el **empleado con salario máximo y ese salario**. El primer for obtiene los distintos puestos de trabajo y el segundo for obtiene los empleados que tienen ese puesto de trabajo. En el if se pregunta si el salario del empleado es el salario máximo de los empleados del oficio del primer for.

```
for $pue in distinct-values(/universidad/departamento/empleado/puesto)
for $emp in /universidad/departamento/empleado[puesto=$pue]
let $sal:=$emp/@salario
let $nom:=$emp/@nombre
return
if($sal = max(/universidad/departamento/empleado[puesto=$pue]/@salario))
then
  <puesto>{data($pue)}
  <maxsalario>{data($sal)}</maxsalario><emple>{data($nom)}</emple>
  </puesto>
else()
```

```
<puesto>Asociado
  <maxsalario>2200</maxsalario>
  <emple/>
</puesto>
<puesto>Profesor
  <maxsalario>2300</maxsalario>
  <emple/>
</puesto>
<puesto>Profesor
  <maxsalario>2300</maxsalario>
  <emple/>
</puesto>
<puesto>Técnico
  <maxsalario>1900</maxsalario>
  <emple/>
</puesto>
<puesto>Tutor
  <maxsalario>2500</maxsalario>
  <emple/>
</puesto>
```

También se podría haber resuelto con un solo for:

```
for $emp in (/universidad/departamento/empleado)
let $pue:=$emp/puesto
let $sal:=$emp/@salario
let $nom:=$emp/@nombre
order by $pue
return
if($sal = max(/universidad/departamento/empleado[puesto=$pue]/@salario))
then
  <puesto>{data($pue)}
  <maxsalario>{data($sal)}</maxsalario><emple>{data($nom)}</emple>
</puesto>
else()
```

```
<puesto>Asociado
  <maxsalario>2200</maxsalario>
  <emple/>
</puesto>
<puesto>Profesor
  <maxsalario>2300</maxsalario>
  <emple/>
</puesto>
<puesto>Profesor
  <maxsalario>2300</maxsalario>
  <emple/>
</puesto>
<puesto>Tutor
  <maxsalario>2500</maxsalario>
  <emple/>
</puesto>
<puesto>Técnico
  <maxsalario>1900</maxsalario>
  <emple/>
</puesto>
```

La consulta a) del apartado anterior, joins de documentos, también se podría haber resuelto con dos for y where.

```
for $emp in (/EMPLEADOS/EMP_ROW)
for $dep in /departamentos/DEP_ROW
let $emple:=$emp/APELLIDO
let $d:=$emp/DEPT_NO
where data($d)= data($dep/DEPT_NO)
return <res>{$emple,$d}{$dep/DNOMBRE}</res>

  <res>
    <APELLIDO>SANCHEZ</APELLIDO>
    <DEPT_NO>20</DEPT_NO>
    <DNOMBRE>INVESTIGACION</DNOMBRE>
  </res>
  <res>
    <APELLIDO>ARROYO</APELLIDO>
    <DEPT_NO>30</DEPT_NO>
    <DNOMBRE>VENTAS</DNOMBRE>
  </res>
  <res>
    <APELLIDO>SALA</APELLIDO>
    <DEPT_NO>30</DEPT_NO>
    <DNOMBRE>VENTAS</DNOMBRE>
  </res>
  <res>
    <APELLIDO>JIMENEZ</APELLIDO>
    <DEPT_NO>20</DEPT_NO>
    <DNOMBRE>INVESTIGACION</DNOMBRE>
  </res>
```

Sentencias de actualización de eXists

- Estas sentencias permiten hacer altas, bajas y modificaciones de nodos y elementos en documentos xml.
- Se pueden usar sentencias de actualización en cualquier punto, pero si se utilizan en la clausula RETURN de una sentencia FLWOR, el efecto de la actualización es inmediato.
- Todas las sentencias de actualización comienzan con la palabra **update** y a continuación la instrucción.

- **INSERT**, Se utiliza para insertar nodos. El lugar de inserción se especifica con:

into (el contenido se añade como último hijo de los nodos especificados)

following (el contenido se añade inmediatamente después de los nodos especificados)

preceding (el contenido se añade antes de los nodos especificados)

El formato es:

Update insert ELEMENTO into EXPRESION Update insert ELEMENTO following EXPRESION Update insert ELEMENTO preceding EXPRESION
--

Ejemplos:

a) Insertar una zona en zonas.xml en la última posición

update insert

```
<zona><cod_zona>50</cod_zona>  
<nombre>Madrid-OESTE</nombre>  
<director>Alicia Ramos Martín</director></zona>  
into /zonas
```

compruebo si se ha insertado

//zona

```
<zona>  
  <cod_zona>20</cod_zona>  
  <nombre>Extremadura-Galicia</nombre>  
  <director>Alicia Pérez</director>  
</zona>  
<zona>  
  <cod_zona>30</cod_zona>  
  <nombre>Levante-Cataluña</nombre>  
  <director>Carlos Grau</director>  
</zona>  
<zona>  
  <cod_zona>40</cod_zona>  
  <nombre>Andalucía</nombre>  
  <director>Rocío Gil</director>  
</zona>  
<zona>  
  <cod_zona>50</cod_zona>  
  <nombre>Madrid-OESTE</nombre>  
  <director>Alicia Ramos Martín</director>  
</zona>  
</zonas>
```

- b) Insertar una cuenta en el documento sucursales.xml del tipo PENSIONES a la sucursal SUC1.

update insert

```
< cuenta tipo="PENSIONES">< nombre>Alberto  
Morales</ nombre>< numero>30302900</ numero>  
< aportacion>5000</ aportacion></ cuenta>  
into /sucursales/sucursal[@codigo="SUC1"]  
//sucursal[@codigo="SUC1"]
```

Send

Clear

Check

Found 1 in 0 seconds.

<<

1

<sucursal telefono="112233" codigo="SUC1">
 <director>Alicia Gómez</director>
 <poblacion>Madrid</poblacion>
 <cuenta tipo="AHORRO">
 <nombre>Antonio García</nombre>
 <numero>123456</numero>
 <saldohaber>21000</saldohaber>
 <saldodebe>200</saldodebe>
 </cuenta>
 <cuenta tipo="AHORRO">
 <nombre>Pedro Gómez</nombre>
 <numero>1111456</numero>
 <saldohaber>12000</saldohaber>
 <saldodebe>0</saldodebe>
 </cuenta>
 <cuenta tipo="AHORRO">
 <nombre>Alicia Martín</nombre>
 <numero>223344</numero>
 <saldohaber>13000</saldohaber>
 <saldodebe>0</saldodebe>
 </cuenta>

 <nombre>María Montes</nombre>
 <numero>667788</numero>
 <saldohaber>3000</saldohaber>
 <saldodebe>100</saldodebe>
 </cuenta>
 <cuenta tipo="PENSIONES">
 <nombre>María Montes</nombre>
 <numero>99667788</numero>
 <aportacion>23000</aportacion>
 </cuenta>
 <cuenta tipo="PENSIONES">
 <nombre>Juan Gil</nombre>
 <numero>99112200</numero>
 <aportacion>30000</aportacion>
 </cuenta>
 <cuenta tipo="PENSIONES">
 <nombre>Alberto Morales</nombre>
 <numero>30302900</numero>
 <aportacion>5000</aportacion>
 </cuenta>
</sucursal>

- **REPLACE**, sustituye el nodo especificado en NODO con VALOR NUEVO.
NODO debe devolver un único ítem: si es un elemento, VALOR_NUEVO debe ser también un elemento.
Si es un nodo de texto o atributo su valor será actualizado con la concatenación de todos los valores de VALOR_NUEVO

Update replace NODO with VALOR_NUEVO

Ejemplos:

- c) Cambia la etiqueta director de la zona 50 y su contenido, en el documento zonas.xml por directora

update replace

/zonas/zona[cod_zona=50]/director

with <directora> Pilar Martin Ramos </directora>

COMPRUEBO SE HA MODIFICADO:

```
/zonas
```

Send Clear Check

Found 1 in 0 seconds.

```
<<
  </zona>
  <zona>
    <cod_zona>20</cod_zona>
    <nombre>Extremadura-Galicia</nombre>
    <director>Alicia Pérez</director>
  </zona>
  <zona>
    <cod_zona>30</cod_zona>
    <nombre>Levante-Cataluña</nombre>
    <director>Carlos Grau</director>
  </zona>
  <zona>
    <cod_zona>40</cod_zona>
    <nombre>Andalucía</nombre>
    <director>Rocío Gil</director>
  </zona>
  <zona>
    <cod_zona>50</cod_zona>
    <nombre>Madrid-OESTE</nombre>
    <directora> Pilar Martin Ramos </directora>
  </zona>
```


- d) Cambia el nodo completo DEP_ROW del departamento 10, por los nuevos datos y las etiquetas que escribamos

**update replace /departamentos/DEP_ROW[DEPT_NO=10]
with**

**<DEPT_ROW><DEPT_NO>10</DEPT_NO>
<DNOMBRE>NUEVO10</DNOMBRE>
<LOC>TALAVERA</LOC></DEPT_ROW>**

```
/departamentos
```

Found 1 in 0 seconds.

<<

1 <departamentos>
 <TITULO>DATOS DE LA TABLA DEPART</TITULO>
 <DEPT_ROW>
 <DEPT_NO>10</DEPT_NO>
 <DNOMBRE>NUEVO10</DNOMBRE>
 <LOC>TALAVERA</LOC>
 </DEPT_ROW>
 <DEPT_ROW>
 <DEPT_NO>20</DEPT_NO>
 <DNOMBRE>INVESTIGACION</DNOMBRE>
 <LOC>MADRID</LOC>
 </DEPT_ROW>
 <DEPT_ROW>
 <DEPT_NO>30</DEPT_NO>
 <DNOMBRE>VENTAS</DNOMBRE>
 <LOC>BARCELONA</LOC>
 </DEPT_ROW>
 <DEPT_ROW>
 <DEPT_NO>40</DEPT_NO>
 <DNOMBRE>PRODUCCION</DNOMBRE>
 <LOC>BILBAO</LOC>

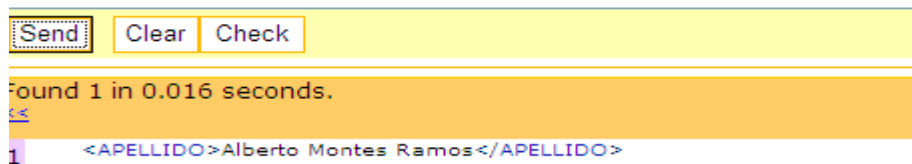
- **VALUE**, actualiza el valor del Nodo especificado en NODO con VALOR_NUEVO. Si NODO es un nodo de texto o atributo su valor será actualizado con la concatenación de todos los valores de VALOR_NUEVO

Update value NODO with 'VALOR_NUEVO'

- e) Cambia el apellido del empleado 7369 del documento empleados.xml por Alberto Montes Ramos
update value /EMPLEADOS/EMP_ROW[EMP_NO=7369]/APELLIDO with 'Alberto Montes Ramos'

compruebo:

```
/EMPLEADOS/EMP_ROW[EMP_NO=7369]/APELLIDO
```

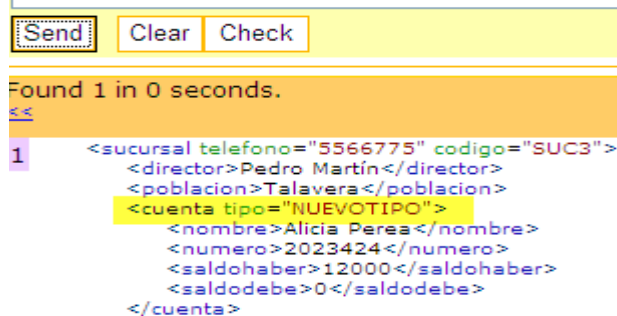


```
Send Clear Check
Found 1 in 0.016 seconds.
<<
1 <APELLIDO>Alberto Montes Ramos</APELLIDO>
```

- f) Cambia el atributo tipo de la primera cuenta de la sucursal SUC3 por NUEVOTIPO, del documento sucursales.xml

update value /sucursales/sucursal[@codigo='SUC3'] /cuenta[1]/@tipo with 'NUEVOTIPO'

```
/sucursales/sucursal[@codigo='SUC3']
```



```
Send Clear Check
Found 1 in 0 seconds.
<<
1 <sucursal telefono="5566775" codigo="SUC3">
  <director>Pedro Martín</director>
  <poblacion>Talavera</poblacion>
  <cuenta tipo="NUEVOTIPO">
    <nombre>Alicia Perea</nombre>
    <numero>2023424</numero>
    <saldohaber>12000</saldohaber>
    <saldodebe>0</saldodebe>
  </cuenta>
  ..
</sucursal>
```

- g) Cambia el salario de los empleados del departamento 10, del documento empleados.xml, subirles 200.

```
for $em in /EMPLEADOS/EMP_ROW[DEPT_NO=10]
let $sal:=$em/SALARIO
return update value $em/SALARIO
with data($sal)+200
```

/EMPLEADOS/EMP_ROW[DEPT_NO=10]

ANTES

Send Clear Check

Found 3 in 0.015 seconds.

<<

```
1 <EMP_ROW>
  <EMP_NO>7782</EMP_NO>
  <APELLIDO>CEREZO</APELLIDO>
  <OFICIO>DIRECTOR</OFICIO>
  <DIR>7839</DIR>
  <FECHA_ALT>1991-06-09</FECHA_ALT>
  <SALARIO>2885</SALARIO>
  <DEPT_NO>10</DEPT_NO>
</EMP_ROW>
2 <EMP_ROW>
  <EMP_NO>7839</EMP_NO>
  <APELLIDO>REY</APELLIDO>
  <OFICIO>PRESIDENTE</OFICIO>
  <FECHA_ALT>1991-11-17</FECHA_ALT>
  <SALARIO>4100</SALARIO>
  <DEPT_NO>10</DEPT_NO>
</EMP_ROW>
3 <EMP_ROW>
  <EMP_NO>7934</EMP_NO>
  <APELLIDO>MUÑOZ</APELLIDO>
  <OFICIO>EMPLEADO</OFICIO>
```

/EMPLEADOS/EMP_ROW[DEPT_NO=10]

**DESPUES DE LA
SUBIDA**

Send Clear Check

Found 3 in 0 seconds.

<<

```
1 <EMP_ROW>
  <EMP_NO>7782</EMP_NO>
  <APELLIDO>CEREZO</APELLIDO>
  <OFICIO>DIRECTOR</OFICIO>
  <DIR>7839</DIR>
  <FECHA_ALT>1991-06-09</FECHA_ALT>
  <SALARIO>3085</SALARIO>
  <DEPT_NO>10</DEPT_NO>
</EMP_ROW>
2 <EMP_ROW>
  <EMP_NO>7839</EMP_NO>
  <APELLIDO>REY</APELLIDO>
  <OFICIO>PRESIDENTE</OFICIO>
  <FECHA_ALT>1991-11-17</FECHA_ALT>
  <SALARIO>4300</SALARIO>
  <DEPT_NO>10</DEPT_NO>
</EMP_ROW>
3 <EMP_ROW>
  <EMP_NO>7934</EMP_NO>
  <APELLIDO>MUÑOZ</APELLIDO>
  <OFICIO>EMPLEADO</OFICIO>
```

- **DELETE**, elimina los nodos indicados en la expresión `expr`

Update delete expr

- h) Elimina la zona con código 50, en el documento `zonas.xml`

update delete /zonas/zona[cod_zona=50]

/zonas

ANTES

Send Clear Check

Found 1 in 0 seconds.

```
<<
<zona>
  <cod_zona>20</cod_zona>
  <nombre>Extremadura-Galicia</nombre>
  <director>Alicia Pérez</director>
</zona>
<zona>
  <cod_zona>30</cod_zona>
  <nombre>Levante-Cataluña</nombre>
  <director>Carlos Grau</director>
</zona>
<zona>
  <cod_zona>40</cod_zona>
  <nombre>Andalucía</nombre>
  <director>Rocío Gil</director>
</zona>
<zona>
  <cod_zona>50</cod_zona>
  <nombre>Madrid-OESTE</nombre>
  <direccion>Alicia Ramos Martín</direccion>
</zona>
</zonas>
```

/zonas

DESPUES DE ELIMINAR

Send Clear Check

Found 1 in 0 seconds.

```
<<
<zona>
  <cod_zona>10</cod_zona>
  <nombre>Madrid-CENTRO</nombre>
  <director>Pedro Martín</director>
</zona>
<zona>
  <cod_zona>20</cod_zona>
  <nombre>Extremadura-Galicia</nombre>
  <director>Alicia Pérez</director>
</zona>
<zona>
  <cod_zona>30</cod_zona>
  <nombre>Levante-Cataluña</nombre>
  <director>Carlos Grau</director>
</zona>
<zona>
  <cod_zona>40</cod_zona>
  <nombre>Andalucía</nombre>
  <director>Rocío Gil</director>
</zona>
</zonas>
```

- **RENAME**, renombra los nodos devueltos en NODO (debe devolver una relación de nodos o atributos) por el NUEVO_NOMBRE

Update rename NODO as NUEVO_NOMBRE

- Cambia de nombre el nodo EMP_ROW del documento empleados.xml

update rename /EMPLEADOS/EMP_ROW as 'fila_emple'

```

/EMPLEADOS

[Send] [Clear] [Check]

Found 1 in 0 seconds.
<<
1  <EMPLEADOS>
    <TITULO>DATOS DE LA TABLA EMPL</TITULO>
    <EMP_ROW>
        <EMP_NO>7369</EMP_NO>
        <APELLIDO>Alberto Montes Ramos</APELLIDO>
        <OFICIO>EMPLEADO</OFICIO>
        <DIR>7902</DIR>
        <FECHA_ALT>1990-12-17</FECHA_ALT>
        <SALARIO>1040</SALARIO>
        <DEPT_NO>20</DEPT_NO>
    </EMP_ROW>
    <EMP_ROW>
        <EMP_NO>7499</EMP_NO>
        <APELLIDO>ARROYO</APELLIDO>
        <OFICIO>VENDEDOR</OFICIO>
        <DIR>7698</DIR>
        <FECHA_ALT>1990-02-20</FECHA_ALT>
        <SALARIO>1500</SALARIO>
        <COMISION>390</COMISION>
        <DEPT_NO>30</DEPT_NO>
    </EMP_ROW>

```

```

/EMPLEADOS

[Send] [Clear] [Check]

Found 1 in 0 seconds.
<<
1  <EMPLEADOS>
    <TITULO>DATOS DE LA TABLA EMPL</TITULO>
    <fila_emple>
        <EMP_NO>7369</EMP_NO>
        <APELLIDO>Alberto Montes Ramos</APELLIDO>
        <OFICIO>EMPLEADO</OFICIO>
        <DIR>7902</DIR>
        <FECHA_ALT>1990-12-17</FECHA_ALT>
        <SALARIO>1040</SALARIO>
        <DEPT_NO>20</DEPT_NO>
    </fila_emple>
    <fila_emple>
        <EMP_NO>7499</EMP_NO>
        <APELLIDO>ARROYO</APELLIDO>
        <OFICIO>VENDEDOR</OFICIO>
        <DIR>7698</DIR>
        <FECHA_ALT>1990-02-20</FECHA_ALT>
        <SALARIO>1500</SALARIO>
        <COMISION>390</COMISION>
        <DEPT_NO>30</DEPT_NO>
    </fila_emple>

```

API XQJ

La Api es una propuesta de estandarización de interfaz java para el acceso a bases de datos XML nativas basado en el modelo de datos XQuery. El objetivo es conseguir un método sencillo y estable de acceso a bases de datos XML nativos.

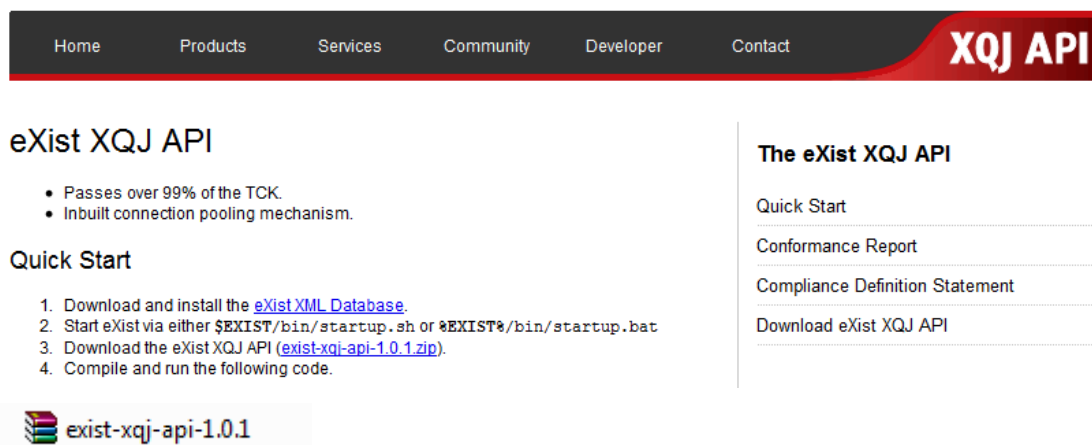
Este estándar cumple los siguientes objetivos:

- Independiente del fabricante
- Soporta el estándar XQuery 1.0
- Facilidad de uso
- Potenciar el uso de esta tecnología en la comunidad java

Al igual que JDBC la filosofía gira en torno al origen de datos y la conexión a este.

Para descargar la API accederemos a la URL:

<http://xqj.net/exist>



Necesitamos importar:

```
import javax.xml.xquery.*;  
import javax.xml.namespace.QName;  
import net.xqj.exist.ExistXQDataSource;
```

Pasos:

1) Configuración de una conexión

- **XQDataSource:** identifica el origen de datos a partir del cual se va a crear la conexión. Cada implementación definirá las propiedades necesarias para efectuar la conexión.

Ejemplo de conexión con eXist:

```
XQDataSource server = new ExistXQDataSource();  
server.setProperty("serverName", "localhost");  
server.setProperty("port", "8080");  
server.setProperty("user", "admin");  
server.setProperty("password", "admin");
```

- **XQConnection:** representa una sesión con la base de datos, manteniendo información de estado, transacciones, expresiones

```
XQConnection conn = server.getConnection();
```

Se cierra la conexión

```
Conn.close()
```

2) Procesar los resultados de una consulta

- **XQPreparedExpression:** objeto creado a partir de una conexión para la ejecución de una expresión múltiples veces. Devuelve un **XQResultSequence** con los datos obtenidos. Igual que en XQExpression, la ejecución se produce llamando al método **executeQuery** y se evalúa teniendo en cuenta el contexto estático en vigor.
- **XQResultSequence:** resultado de la ejecución de una sentencia; contiene un conjunto de 0 o más **XQResultItem**. Es un objeto recorrible.

Ejemplo que obtiene los empleados del departamento 10, utilizaremos `getItemAsString` para que devuelva los elementos como cadenas

```
XQPreparedExpression consulta;  
XQResultSequence resultado;  
Consulta =  
con.prepareExpression("/EMPLEADOS/EMP_ROW[DEPT_NO=10]");  
Resultado = consulta.executeQuery();  
While (resultado.next())  
    System.out.println("Elemento: "+resultado.getItemAsString(null));
```

- **XQResultItem:** Objeto que representa un elemento de un resultado, inmutable, válido hasta que se llame al método `close` suyo o de la `XQResultSequence` a la que pertenece. En el ejemplo anterior lo podemos poner así:

```
Consulta =  
con.prepareExpression("/EMPLEADOS/EMP_ROW[DEPT_NO=10]");  
Resultado = consulta.executeQuery();  
XQResultItem r_item;  
While (resultado.next()){  
    r_item=(XQResultItem) resultado.getItem();  
    System.out.println("Elemento:"+r_item.getItemAsString(null));  
}
```

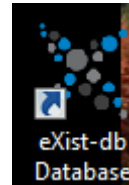
- **XQItem:** representación de un elemento en XQuery, es inmutable y una vez creado su estado interno no cambia
- **XQSequence:** representación de una secuencia del modelo de datos XQuery, contiene un conjunto de 0 o más `XQItem`. Es un objeto recorrible.

Con XQJ no necesitamos seleccionar la colección de documentos XML, la búsqueda la realiza en todas la colecciones. Tampoco admite el uso de sentencias de actualización de `eXist`.

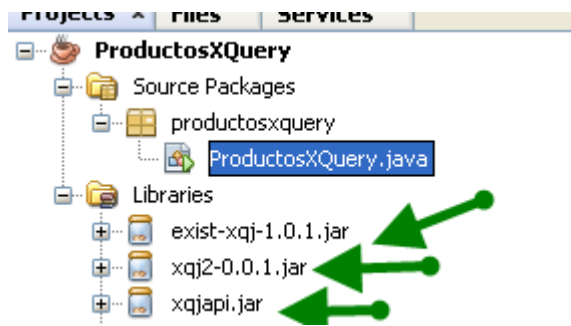
____Ejemplo para mostrar los datos del documento productos.xml, si tenemos productos en distintos documentos se mostrarán las etiquetas de todos los documentos.

PRUEBALO EN NETBEANS

Pasos:



- Arrancar la base de datos eXist
- Crear un proyecto en Netbeans llamado ProductosXQuery
- Añadir a las librerías de la API XQJ al proyecto.



```
package productosxquery;
import javax.xml.xquery.*;
import net.xqj.exist.ExistXQDataSource;

public class ProductosXQuery {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) throws Exception {
        // TODO code application logic here
        XQDataSource server=new ExistXQDataSource();
        server.setProperty("serverName", "localhost");
        server.setProperty("port", "8080");
        server.setProperty("user","admin");
        server.setProperty("password","admin");
        XQPreparedExpression consulta;
        XQResultSequence resultado;
        XQConnection conn=server.getConnection();

        System.out.println("-----consulta documentos productos.xml-----");
        consulta=conn.prepareExpression("for $de in /productos return $de");
        resultado=consulta.executeQuery();
        while(resultado.next()){
            System.out.println("Elemento "+resultado.getItemAsString(null));
        }
        //la conexion se tiene que cerrar fuera del bucle while
        //si no se hace, te visualizará los datos pero te dará un XQException al final
        conn.close();
    }
}
```

```

run:
-----consulta documentos productos.xml-----
Elemento <productos>
  <TITULO xmlns="">DATOS DE LA TABLA PRODUCTOS</TITULO>
  <produc xmlns="">
    <cod_prod xmlns="">1010</cod_prod>
    <denominacion xmlns="">Placa Base MSI G41M-P26</denominacion>
    <precio xmlns="">50</precio>
    <stock_actual xmlns="">10</stock_actual>
    <stock_minimo xmlns="">3</stock_minimo>
    <cod_zona xmlns="">10</cod_zona>
  </produc>
  <produc xmlns="">
    <cod_prod xmlns="">1011</cod_prod>
    <denominacion xmlns="">Micro Intel Core i5-2320</denominacion>
    <precio xmlns="">120</precio>
    <stock_actual xmlns="">3</stock_actual>
    <stock_minimo xmlns="">5</stock_minimo>
    <cod_zona xmlns="">10</cod_zona>
  </produc>
  <produc xmlns="">
    <cod_prod xmlns="">1012</cod_prod>
    <denominacion xmlns="">Micro Intel Core i5 2500</denominacion>
    <precio xmlns="">170</precio>
    <stock_actual xmlns="">5</stock_actual>
    <stock_minimo xmlns="">6</stock_minimo>
    <cod_zona xmlns="">20</cod_zona>
  </produc>

```