



IES
Puerto
de la
Cruz
Telesforo Bravo

CICLO FORMATIVO DE GRADO SUPERIOR:

“DESARROLLO DE APLICACIONES WEB”



MÓDULO:

**Lenguajes de marcas y
sistemas de gestión de
información**

(4 horas semanales)





Índice

TEMA 10.- XQUERY.	3
1.- INTRODUCCIÓN A XQUERY.	3
2.- TECNOLOGÍAS RELACIONADAS.	4
3.- LA CONSULTA MÁS BÁSICA EN XQUERY.	5
4.- LAS CONSULTAS FLWOR.	5
4.1.- Diferencias entre FOR y LET.	8
4.2.- Variables posicionales en la cláusula FOR.	9
5.- TRANSFORMANDO EL XML CON XQUERY.	9
6.- FUNCIONES XQUERY.	11
7.- CUANTIFICADORES.	13
8.- JOINS.	17
8.1.- Inner & outer joins.	18
9.- CUESTIONES FINALES.	20
9.1.- Ordenamiento.	20
9.2.- Comentarios.	20
9.3.- Comentario final.	21



TEMA 10.- XQuery.

1.- Introducción a XQuery.


Actualmente, XML se ha convertido en una herramienta de uso cotidiano en los entornos de tratamiento de la información. El uso de esta tecnología y sus tecnologías derivadas aumenta rápidamente debido a que es una manera relativamente sencilla de estructurar y compartir información entre plataformas diferentes.

Sin embargo ante proyectos complejos, con mucha información, nos encontramos con que es problemático realizar búsquedas o consultas **entre varios ficheros** de una manera sencilla. Para estos menesteres muchos programadores realizaban programas de código cuya principal función es tomar una fuente de datos XML y analizarla buscando la información deseada. En este sentido se desarrollaron modelos trabajo sobre el árbol sintáctico, llamados **parsers**, de los que son ejemplos SAX y DOM. Sin embargo requieren escribir bastante código en un proyecto de cierta complejidad.

Ante esta situación el W3C comenzó el desarrollo del lenguaje **XQuery**, cuyo principal objetivo es seleccionar los elementos de interés de información en XML, reorganizándola y posiblemente transformándola, retornando los resultados en una estructura que el usuario desee.

¿Qué nos permite hacer XQuery?

- Seleccionar información basada en un criterio específico.
- Buscar información en un documento o conjunto de documentos.
- Unir datos desde **múltiples documentos** o colección de documentos.
- Organizar, agrupar y resumir datos
- Transformar y reestructurar datos XML en otro vocabulario o estructura.
- Desempeñar cálculos aritméticos sobre números y fechas.
- Manipular cadenas de caracteres y trabajar con tipos de datos.



XQuery es a XML lo mismo que **SQL** es a las bases de datos relacionales: una forma rápida, compacta y estructurada de realizar consultas sobre colecciones de datos.

XQuery es un lenguaje independiente de la plataforma.

XQuery es una recomendación del W3c.



Existen bases de datos nativas en XML, es decir, almacenan la información directamente en archivos XML. Este tipo de bases de datos, antes poco utilizadas, se están haciendo con cuota de mercado. Ejemplos son: **Berkeley DB XML**, **eXist**, etc.

Por otra parte muchas bases de datos relacionales de amplio uso soportan desde hace algunas versiones tanto XML como XQuery. Es el caso de **Oracle 10g**, **IBM DB2 v9** y **Microsoft SQL-Server 2005**.

Y por último también son cada vez más las aplicaciones de uso muy diverso, no embebidas dentro de una base de datos sino con funcionamiento independiente que trabajan con la información estructurada en ficheros XML.

2.- Tecnologías relacionadas.

XQuery está relacionado con otras tecnologías:

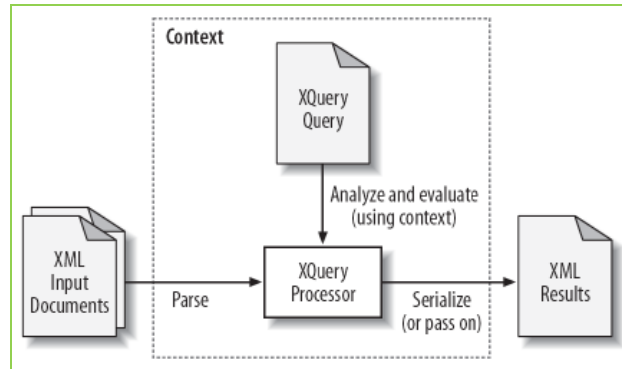
XPath y XSLT: ya los hemos utilizado en temas anteriores. Utilizando estas dos tecnologías podemos acceder a información (usando XPath 1.0) y transformar un documento de entrada en otro de salida (usando XSLT). Bien, XQuery incluye el uso de la versión **XPath 2.0**, que soporta las consultas FLWORS y los Constructores XML (no existentes en la versión 1.0 de XPath). XQuery también es capaz de utilizar varios documentos realizando consultas en los mismos (algo parecido a los JOIN de SQL cuando trabajamos con varias tablas). Esta es una de las principales ventajas con respecto al XSLT.

SQL: la mayor parte de las ideas y construcciones sintácticas del XQuery parten de la tecnología SQL. Actualmente estos dos estándares conviven en muchas bases de datos. Las bases de datos relacionales se utilizan para trabajar con datos altamente estructurados. En algunos casos XQuery permite trabajar con datos cuyo nivel de estructuración es menos exigente.

XSD (Xml Schema): el XQuery utiliza todos los tipos de datos que se definen en XSD (recuerda algunos como boolean, date, decimal, string, etc). Si el documento XML que estamos utilizando tiene asociado un esquema entonces podremos utilizar los tipos de datos en nuestras consultas XQuery y realizar conversiones de tipos. Si por el contrario el documento de entrada no tiene ningún esquema definido no podremos utilizar los tipos de datos.

Namespaces: el XQuery soporta también la definición y uso de los espacios de nombres.

La forma de funcionamiento de esta tecnología es bastante similar a la del estándar XSLT que ya hemos aprendido. Tomando un fichero de entrada en XML y un fichero que define consultas XQuery se aplicará un proceso de “parseo” que provocará como resultado un fichero de salida diferente. El esquema básico de procesamiento se muestra en la siguiente figura:



3.- La consulta más básica en XQuery.

Dado que XPath 2.0 es en realidad un subconjunto del estándar XQuery, la consulta más básica que podemos realizar es simplemente la ejecución de una expresión XPath sobre un fichero escrito en XML. La sintaxis para estas consultas es la ya conocida sintaxis de XPath, con sus ejes, @ para atributos, corchetes [] para predicados, etc.

Para realizar una consulta de este estilo simplemente tendremos que indicar qué fichero queremos abrir. Esto se hace con **doc**("...fichero ..."). Observa el ejemplo:

```
doc("alumnos.xml")/instituto/curso[@nombre = "1º ASIR"]/alumno/nombre
```

Esta consulta selecciona todos los nombres de los alumnos de 1º de ASIR, del documento alumnos.xml. Pero aparte de seleccionar los nombres no hacemos nada más, no podemos manipular, ni transformar la información. Para eso usamos las consultas FLWOR.

4.- Las consultas FLWOR.

Antes de presentar las consultas FLWOR es necesario recordar un detalle importante: a diferencia de lo que sucede en SQL, en XQuery las expresiones y los valores que devuelven son **dependientes del contexto** (recuerda que estamos trabajando sobre un árbol sintáctico). Por ejemplo: los nodos que aparecerán en el resultado dependen de los namespaces, de la posición que ocupe en el fichero de datos, del nodo padre, etc.

En XQuery las consultas pueden estar compuestas por cláusulas de hasta cinco tipos distintos. Las consultas siguen la norma **FLWOR** (que se lee **flower**).

FLWOR son las siglas de:

- For, Let, Where, Order, Return.



Veamos un ejemplo:

```
xquery version "1.0";
declare default element namespace "http://misitio.com";
declare copy-namespaces no-preserve, no-inherit;

<repetidores>
{
  for $variable in /instituto/curso/alumno
  where $variable/repetidor = "true"
  order by $variable/apellidos
  return $variable/apellidos
}
</repetidores>
```

Darí­a como resultado un fichero XML con los apellidos de los alumnos que son repetidores (ordenados ascendentemente por los apellidos):

```
<?xml version="1.0" encoding="UTF-8"?>
<repetidores xmlns="http://misitio.com">
  <apellidos>Domí­nguez</apellidos>
  <apellidos>Hernández Hernández</apellidos>
  <apellidos>Álvarez Pérez</apellidos>
</repetidores>
```

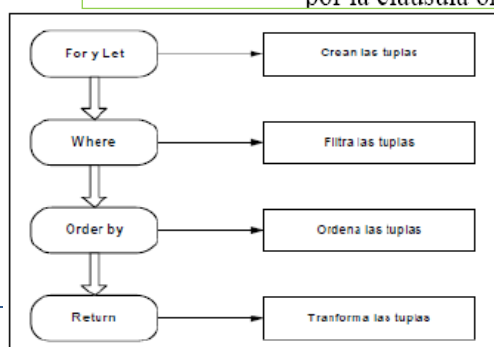
Nota: para probar este ejemplo en Editix hemos quitado de la consulta la apertura del documento con **doc("...")**. Después en el menú XSLT/XQuery seleccionaremos la opción “Transform a document with this XQuery request...” → y ahí indicaremos qué fichero tiene los datos de entrada.



PRÁCTICA 01.- Crea el fichero practica01.xql y copia en él el código del ejemplo propuesto. Genera el fichero resultado01.xml en el Editix.

Analicemos cada uno de los apartados de una consulta FLWOR:

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:** establece una iteración en cada uno de los nodos del resultado de la expresión XPath que le acompaña.

```
for $mivble in doc("alumnos.xml")/instituto/curso[@nombre="1º ASIR"]/alumno
```

Este ejemplo realiza una iteración (similar a <xsl:for-each> en XSLT) por cada nodo alumno de 1º de ASIR. En cada paso de esta iteración el nodo contexto (cada uno de los nodos alumno en este ejemplo) se asigna a la variable \$mivble.

- **La cláusula Where:** nos permite filtrar y quedarnos con un subconjunto de nodos.

```
where $variable/repetidor = "True"
```

En este ejemplo debemos recordar que \$variable está asignada a cada uno de los nodos alumno seleccionados en el for. De este conjunto de nodos nos quedaremos con aquellos cuyo hijo repetidor tiene valor "True".

- **La cláusula Order by:** establece la ordenación de los resultados en el fichero de salida, cosa que no podemos controlar sólo con XPath.

```
order by $variable/apellidos
```

- **La cláusula Return:** es la encargada de generar el contenido de la salida. Pronto veremos que con esta cláusula es posible modificar el formato del fichero resultante.

```
return $variable/nombre
```

Aquí como resultado de salida sólo obtenemos los nodos nombre (con su contenido incluido).

- **La cláusula LET:** no la hemos utilizado aún. Puede usarse sola (sin cláusula for) o bien junto a un for. Si no existe una cláusula for genera una única tupla o único resultado.

Ejemplo de uso de LET con FOR:

```
for $variable in doc("alumnos.xml")/instituto/curso[@nombre="1º ASIR"]/alumno
let $nombre := $variable/nombre
where $variable/repetidor = "True"
order by $variable/apellidos
return $nombre
```

En este ejemplo la cláusula let sólo sirve para crear la variable \$nombre que contiene el nombre del alumno.



PRÁCTICA 02.- Crea practica02.xql. Utilizando en una cláusula let la variable \$num genera un fichero xml con toda la información de los alumnos que tengan menos de cuatro materias.

Resultado:

```
<?xml version="1.0" encoding="UTF-8"?>
<alumnos xmlns="http://misitio.com">
  <alumno cial="A92R590">
    <nombre>Miguel</nombre>
    <apellidos>Rodríguez</apellidos>
    <repetidor>False</repetidor>
    <notas>
      <asignatura>Programación</asignatura>
      <nota>4</nota>
      <asignatura>Multimedia</asignatura>
      <nota>3</nota>
    </notas>
  </alumno>
</alumnos>
```

4.1.- Diferencias entre FOR y LET.

Lo primero que debemos aclarar es que **ninguna** de las cláusulas FLWOR es obligatoria. Para comprender las diferencias entre FOR y LET veamos un ejemplo:

```
<alumnos>
{
for $n in //nombre
return <datos>{$n}</datos>
}
</alumnos>
```

Arroja el siguiente resultado:

```
<alumnos xmlns="http://misitio.com">
  <datos>
    <nombre>Javier</nombre>
  </datos>
  <datos>
    <nombre>María</nombre>
  </datos>
  <datos>
    <nombre>Iván</nombre>
  </datos>
  ...
</alumnos>
```

Observa el uso de llaves en la cláusula return. **Se genera el contenido de return para cada valor de la variable \$n.** Ahora vemos el resultado de la misma consulta si en lugar de usar for usamos un let:

```
<alumnos>
{
let $n := //nombre
return <datos>{$n}</datos>
}
</alumnos>
```

Arroja el siguiente resultado:

```
<?xml version="1.0" encoding="UTF-8"?>
```



```
<alumnos>
  <datos>
    <nombre>Javier</nombre>
    <nombre>María</nombre>
    <nombre>Iván</nombre>
    <nombre>Marcos</nombre>
    <nombre>Luis Fernando</nombre>
    <nombre>Elena</nombre>
    <nombre>Irina</nombre>
    <nombre>Genaro</nombre>
    <nombre>Miguel</nombre>
  </datos>
</alumnos>
```

Observamos varias cosas. La primera es que let usa el operador de asignación `:=` mientras que for usa la palabra reservada `"in"`. La segunda cuestión es que el contenido de return no se ejecuta completo para cada alumno: el elemento `<datos>` aparece una única vez. LET no es un bucle, es sólo una asignación de valor. La variable `$n` tiene el valor formado por el conjunto de nodos nombre del árbol sintáctico de entrada.

La cláusula for por el contrario provoca una iteración. En cada paso de esta iteración se asigna un valor de entre el conjunto de nodos resultado a la variable `$n` y se genera el contenido de result.

4.2.- Variables posicionales en la cláusula FOR.

Cuando estamos realizando una iteración mediante una cláusula FOR nos puede interesar fijar en una variable adicional la posición de cada elemento seleccionado. Para esto se usan las variables posicionales, que utilizan la **palabra reservada at antes de la variable**. Prueba el siguiente ejemplo, que usa una variable posicional llamada `$i`:

```
xquery version "1.0";
<alumnos>
{
  for $a at $i in doc("alumnos.xml")//alumno
  return <alumno numero="{ $i }">{ $a/nombre }</alumno>
}
</alumnos>
```

5.- Transformando el XML con XQuery.

El XQuery, al igual que el XSLT, permite realizar transformaciones de datos XML. Para ello es importante fijarse en el uso de las llaves `{ }`. En XQuery las llaves indican que el código que está dentro debe **evaluarse**. Veamos varios casos de transformación de documentos.

```
xquery version "1.0";
<html><head><title>Lista</title></head>
<body>
<h3>Hay { count(doc("alumnos.xml")/instituto/curso/alumno) } alumnos</h3>
<ul>
{
  for $a in doc("alumnos.xml")/instituto/curso/alumno
  order by $a/apellidos
```

```

    return <li>{string($a/apellidos)}, {string($a/nombre)}</li>
}
</ul>
</body>
</html>

```

Con este ejemplo estamos creando una lista no ordenada Xhtml con los apellidos y los nombres de los alumnos, ordenados ascendentemente por apellido.

Funciones para acceder a los contenidos de los elementos:

string(elemento): devuelve el contenido de texto del element
 data(elemento): devuelve el contenido de un elemento, si es de un tipo soportado en Xml Schema.

Si en el ejemplo anterior no hubiéramos utilizado string(), es decir, si hacemos sólo return { \$a/apellidos } , { \$a/ nombre } , lo que copiamos en el fichero de salida serán los nodos del árbol. La lista quedaría así:

```

<ul>
  <li><apellidos>Domínguez</apellidos>, <nombre>Javier</nombre></li>
  ...

```

Con el uso de string estamos descartando el elemento y quedándonos sólo con el contenido. Este mecanismo también lo podremos utilizar para crear valores de atributos, modificar nombres de elementos, generar otros ficheros XML, XHTML u otros formatos.



PRÁCTICA 03.- Crea practica03.xql, que genere un fichero XHTML como el siguiente:

```

<html>
  <head>
    <title>Lista</title>
  </head>
  <body>
    <h3>Hay 9 alumnos</h3>
    <ol type="1">
      <li id="A92T861">Marcos del curso 2º ASIR</li>
      <li id="A92R543">Genaro del curso 1º DAI</li>
      <li id="A92R003">Luis Fernando del curso 2º ASIR</li>
      <li id="A95M600">Irina del curso 1º DAI</li>
      <li id="A18X111">Javier del curso 1º ASIR</li>
      <li id="A87R122">Elena del curso 2º ASIR</li>
      <li id="A92R590">Miguel del curso 1º DAI</li>
      <li id="A90R112">Iván del curso 1º ASIR</li>
      <li id="A27M242">María del curso 1º ASIR</li>
    </ol>
  </body>
</html>

```

Nota: en el caso de estar formando atributos en principio no es necesario explicitar que queremos el contenido del elemento con string o data, se hace directamente.



PRÁCTICA 04.- Crea practica04.xql, que genere una tabla XHTML como esta:

Hay 9 alumnos

Alumno	Materias aprobadas
Elena	4 materias
Genaro	4 materias
Irina	3 materias
IvÁjn	2 materias
Javier	3 materias
Luis Fernando	0 materias
Marcos	4 materias
MarÁa	4 materias
Miguel	0 materias



PRÁCTICA 05.- Crea practica05.xql, que modifica la práctica anterior para que no aparezcan en la tabla alumnos con menos de tres materias aprobadas:

Hay 9 alumnos

Alumno	Materias aprobadas
Elena	4 materias
Genaro	4 materias
Marcos	4 materias
MarÁa	4 materias

6.- Funciones XQuery.

El estándar XQuery incorpora muchas funciones predefinidas. Éstas permiten manipular cadenas, fechas, tipos de datos simples de Xml Schema, realizar cálculos matemáticos, etc.

Además de esto el estándar XQuery soporta la creación de nuestras propias funciones, así como la creación de librerías que agrupan funciones y podrán ser reutilizadas en otros ficheros.

Todas las funciones podrán ser llamadas en cualquier parte de una consulta. Si la función está en una librería externa en ésta tenemos que declarar un namespace y en nuestro fichero .xql debemos importarlo.

Recuerda que siempre que necesites usar una función debes acompañarla de paréntesis y dentro de éstos los posibles parámetros de la función utilizada.

Numeric functions

abs, avg, ceiling, floor, max, min, number, round, round-half-to-even, sum

String functions

codepoint-equal, codepoints-to-string, compare, concat, contains, default-collation, ends-with, lang, lower-case, matches, normalize-space, normalize-unicode, replace, starts-with, string-join, string-length, string-to-codepoints, substring, substring-after, substring-before, tokenize, translate, upper-case

Date functions

adjust-date-to-timezone, adjust-dateTime-to-timezone, adjust-time-to-timezone, current-date, current-dateTime, current-time, implicit-timezone, dateTime

Date functions (component extraction)

day-from-date, day-from-dateTime, days-from-duration, hours-from-dateTime, hours-from-duration, hours-from-time, minutes-from-dateTime, minutes-from-duration, minutes-from-time, month-from-date, month-from-dateTime, months-from-duration, seconds-from-dateTime, seconds-from-duration, seconds-from-time, timezone-from-date, timezone-from-dateTime, timezone-from-time, year-from-date, year-from-dateTime, years-from-duration

Boolean functions

boolean, false, true, not

Document and URI functions

base-uri, collection, doc, doc-available, document-uri, encode-for-uri, escape-html-uri, iri-to-uri, resolve-uri, root, static-base-uri

Name and namespace functions

QName, in-scope-prefixes, local-name, local-name-from-QName, name, namespace-uri, namespace-uri-for-prefix, namespace-uri-from-QName, node-name, prefix-from-QName, resolve-QName

Node-related functions

data, deep-equal, empty, exists, id, idref, nilled, string

Sequence-related functions

count, distinct-values, index-of, insert-before, last, position, remove, reverse, subsequence, unordered

Error handling and trapping functions

error, exactly-one, one-or-more, trace, zero-or-one



PRÁCTICA 06.- Crea practica06.xql, que utilizando convenientemente la función substring obtenga los dos primeros número dentro del Cial del alumno y cree un documento xml como el siguiente (necesitarás averiguar qué parámetros necesita la función substring para extraer los caracteres 2 y 3 del cial, y antes agregarles la cadena “19”):

Alumnos:

Alumno	Nacio en:
Elena	1987
Genaro	1992
Irina	1995
IvÃ¡n	1990
Javier	1918
Luis Fernando	1992
Marcos	1992
MarÃa	1927
Miguel	1992



PRÁCTICA 07.- Crea practica07.xql, que utilizando convenientemente la función month-from-date muestre las faltas del mes de mayo en el fichero faltas.xml. La tabla a generar es la siguiente:

Faltas en el mes de mayo:

Cial	Fecha:	Hora de clase
A18X111	2011-05-20	3
A18X111	2011-05-20	4
A18X111	2011-05-20	5
A18X111	2011-05-20	6
A27M242	2011-05-08	2
A27M242	2011-05-20	6
A92R590	2011-05-02	1
A92R590	2011-05-02	6
A92R590	2011-05-03	6



PRÁCTICA 08.- Crea practica08.xql, que muestre las faltas injustificadas (aquellas de tipo "I") en el mes de abril de los alumnos nacidos en 1992:

Faltas injustificadas del mes de abril para alumnos del 92:

Cial	Fecha:	Hora de clase
A92R590	2011-04-02	6
A92R590	2011-04-03	6
A92T861	2011-04-20	4



PRÁCTICA 09.- Crea practica09.xql, que muestre la nota media de cada alumno de 1º de ASIR (usando la función avg):

Nota media de los alumnos de 1º de ASIR

Nombre	Apellidos	Nota media
IvÃjn	RodrÃguez PÃrez	6.25
Javier	MÃrquez FernÃndez	6
MarÃa	Ã lvarez PÃrez	5.75



PRÁCTICA 10.- Crea practica10.xql, que muestre las materias de los alumnos cuyo nombre de asignatura termine en "s":

Materias que acaban en -s

Nombre	Apellidos	Materia
Javier	MÃrquez FernÃndez	Lenguajes de marcas
Javier	MÃrquez FernÃndez	Redes
Javier	MÃrquez FernÃndez	Sistemas Operativos
MarÃa	Ã lvarez PÃrez	Lenguajes de marcas
MarÃa	Ã lvarez PÃrez	Redes

7.- Cuantificadores.

Las expresiones cuantificadas permiten detectar si alguno (**some**) o todos (**every**) los elementos seleccionados cumplen una condición particular. Los

cuantificadores existenciales en XQuery usan las palabras reservadas **some**, **every** y **satisfies**.

Some: recupera las tuplas en las que **ALGÚN** nodo cumple la condición descrita.

Every: recupera las tuplas en las que **TODOS** los nodo cumplen la condición descrita.

Ejemplo:

```
some $a in doc("alumnos.xml")//alumno
satisfies ($a/notas/asignatura = "Redes")
```

Esta consulta devolvería true, ya que hay por lo menos algún alumno en el fichero que tiene una materia llamada "Redes".

Ejemplo:

```
every $a in doc("alumnos.xml")//alumno
satisfies ($a/notas/asignatura = "Redes")
```

Esta consulta devolvería false, ya que NO TODOS los alumnos en el fichero tienen una materia llamada "Redes". Sólo obtendríamos true si todos los alumnos tuvieran esta materia.

Las consultas con **some** y **every** devuelven valores true o false.

Hay que resaltar que casi cualquier expresion escrita con cuantificadores tiene una forma equivalente de escritura con una expresión FLOWR o bien como una consulta XPath, pero el código sería mucho más largo.

Por otra parte es muy frecuente usar la función **not(...)** para cambiar el significado de la consulta.

not(some ...) significa "no alguno", o lo que es lo mismo → NINGUNO.

Ejemplo:

```
not (some $a in doc("alumnos.xml")//alumno
satisfies ($a/notas/asignatura = "Canto")
)
```

Esta consulta devolvería "true", ya que NINGÚN alumno en el fichero está matriculado en "Canto".

not(every ...) significa "no todos", o lo que es lo mismo → NINGUNO.

Ejemplo:

```
not (every $a in doc("alumnos.xml")//alumno
satisfies ($a/notas/asignatura = "Canto")
)
```

Esta consulta devolvería "true", ya que NO TODOS los alumnos en el fichero están matriculados en "Canto".

NOTA: estas expresiones cuantificadas pueden usarse dentro de las consultas FLOWR, en especial resultan muy útiles en la cláusula where.



PRÁCTICA 11.- Crea practica11.xql, que muestre los cursos en los que hay algún alumno matriculado en “Prácticas de empresa”:

Cursos en los que se imparte "Prácticas de empresa"

Curso
2Â° ASIR



PRÁCTICA 12.- Crea practica12.xql, que muestre los cursos en los que todos los alumnos estén matriculados en “Lenguajes de marcas”

Cursos en los que todos los alumnos asisten a "Lenguajes de marcas":

Curso
1Â° ASIR



PRÁCTICA 13.- Crea practica13.xql, que muestre los alumnos que tienen todas sus materias aprobadas:

Alumnos con todo aprobado:

Alumno	Apellido
MarÃa	Ã lvarez PÃ©rez
Marcos	DomÃnguez
Elena	RamÃrez
Genaro	HernÃndez HernÃndez



PRÁCTICA 14.- Crea practica14.xql, que muestre los alumnos que no tienen ninguna materia aprobada, indicando el número de materias:

Alumnos con ninguna materia aprobada:

Alumno	Apellido	NÃºmero de materias
Luis Fernando	JimÃ©nez PÃ©rez	4
Miguel	RodrÃguez	2

También es posible enlazar varias variables en una expresión cuantificada, al igual que en cualquier expresión FLOWR. Para ello usaremos la **coma** como elemento separador.

Ejemplo:

some \$i in (1 to 3), \$j in (10, 11)

satisfies \$j - \$i = 7

Aquí estamos preguntando si hay algún valor \$i (entre 1 y 3), y algún valor \$j (ya sea 10 ó 11) tales que \$j - \$i vale 7. Concretamente cuando \$i=3 y \$j=10 se cumple la condición, con lo que la expresión vale true.

Otro cuantificador interesante es **distinct-values**, que funciona de forma similar al **distinct** del SQL. Permite seleccionar de un conjunto de elementos sólo los valores que son atómicamente diferentes.

Ejemplo 1:

Seleccionar del fichero faltas.xml aquellas fechas en las que se han registrado faltas.

```
distinct-values(doc("faltas.xml"))/falta/@fecha
```

Ejemplo 2:

Queremos seleccionar el número de faltas por cada fecha:

```
<html><head><title>Lista</title></head>
<body>
<h3>Días de falta:</h3>
<table cols="2" width="50%" border="2px">
<tr><td>Fecha</td><td>Num. Faltas</td></tr>
{
for $a in distinct-values(doc("faltas.xml"))/falta/@fecha
let $b := count(doc("faltas.xml"))/falta[@fecha = $a]
return <tr><td> { string($a) } </td><td>{ string($b) } </td></tr>
}
</table>
</body>
</html>
```

Hacemos una iteración con la variable \$a, a la que asignamos los **distintos valores** de fecha existentes en el fichero. Para cada uno de esos valores diferentes lo que queremos es contar cuántas veces aparece en el fichero. Esto lo hacemos con un **let** que asigna a la variable \$b la cuenta de los elementos falta cuyo atributo coincida con el de \$a. Por último sacamos los resultados en el result.



PRÁCTICA 15.- Crea practica15.xql, que muestre para cada alumno de comentarios.xml el número de días distintos en los que tiene comentarios y el total de comentarios del alumno:

Comentarios por alumno:

CIAL	Fechas con Comentarios	Numero de Comentarios
A18X111	1	2
A27M242	2	2
A90R112	2	2
A92T861	3	4
A95M600	3	4
A92R590	1	2



PRÁCTICA 16.- Crea `practical16.xql`, que muestre una tabla con el número de comentarios por fecha para cada alumno. Observa la tabla:

Comentarios por fecha:

Fecha	Numero de Comentarios
2011-05-20	5
2010-11-22	4
2011-04-08	7

8.- Joins.

Una de las mayores ventajas del XQuery con respecto al XSLT es la posibilidad de realizar consultas entre distintos documentos de entrada. Mientras que en XSLT tenemos un único documento de entrada y generamos un único documento de salida, en XQuery podemos realizar **joins**, es decir, varios documentos de entrada generarán un único documento de salida.

El concepto de **join** o unión entre tablas es equivalente al realizado en SQL, y se produce realizando un producto cartesiano entre los documentos de entrada.

La idea es que usando el símbolo **coma** en cualquier cláusula FOR o LET de una consulta FLOWR se producirá el producto cartesiano entre las variables declaradas. Analicemos un primer ejemplo:

```
for $alum in doc("alumnos.xml")//alumno,
    $comen in doc("comentarios.xml")//alumno[@cial = $alum/@cial]/comentario
order by $alum/nombre, $comen/@fecha
return
  <tr><td> {string($alum/nombre)}</td>
    <td>{string($comen/@fecha)}</td>
    <td>{string($comen)}</td></tr>
```

→\$alum in doc("alumnos.xml")//alumno → selecciona los alumnos.

→\$comen in doc("comentarios.xml")//alumno[@cial = \$alum/@cial]/comentario

La variable \$comen seleccionará cada comentario con el mismo cial que \$alum. Al encontrarnos dentro de un FOR y fijar esta igualdad en el predicado estamos realizando un producto cartesiano entre los dos ficheros implicados.

Esta es la **forma de hacer un inner join**. El resultado sería una tabla como la que mostramos a continuación (observa que también hemos puesto dos criterios en el ORDER BY, el primero para ordenar por alumno, y el segundo para ordenar por fecha):

Comentarios por fecha:

Alumno	Fecha	Comentario
Irina	2010-11-22	Solicita al departamento fotocopias del libro Redes de Tanembaun
Irina	2011-04-08	Entrega supuesto práctico de subnetting
Irina	2011-04-08	Entrega presupuesto para un CPD en papel
Irina	2011-05-20	Resuelve dudas de sus compañeros en el foro
Iván	2010-11-22	Voluntario para ejercicio en pizarra de Gestión de Procesos
Iván	2011-04-08	Entregado supuesto práctico de subnetting

También podríamos haber puesto la unión del join en una cláusula where, en lugar de en el predicado de la segunda variable. La siguiente consulta es equivalente a la anterior:

```
for $alum in doc("alumnos.xml")//alumno,
    $comen in doc("comentarios.xml")//comentario
where $alum/@cial = $comen/../@cial
order by $alum/nombre, $comen/@fecha
return
<tr><td> {string($alum/nombre)}</td>
<td>{string($comen/@fecha)}</td>
<td>{string($comen)}</td></tr>
```



PRÁCTICA 17.- Crea practica17.xql, que muestre una tabla con las faltas injustificadas (de tipo =”I”) para cada alumno:

Comentarios por fecha:

Alumno	Fecha	Comentario
Iván	I	2010-11-08
Iván	I	2010-11-22
Iván	I	2010-12-18
Javier	I	2011-05-20
Javier	I	2011-05-20
Javier	I	2011-05-20
Javier	I	2011-05-20
Marcos	I	2011-02-03
Marcos	I	2011-04-20
María	I	2011-05-20

8.1.- Inner & outer joins.

Los productos cartesianos que hemos visto hasta ahora se conocen como **inner joins** o “uniones internas”. Si al unir los dos ficheros xml se observa, por ejemplo, que un alumno no tiene faltas entonces no aparece en la consulta de salida. Sólo aparece aquella información que esté presente en las dos o más fuentes de datos de entrada.

Los **outer joins** o “uniones externas” por el contrario mostrarán toda la información de las fuentes de entrada. Basta con que el dato esté en una de las fuentes para formar parte de los resultados de salida. De esta forma si un alumno, por ejemplo, no tiene faltas, obtendremos una fila de la tabla con el nombre del alumno, y aparecerá vacía su información de faltas (pues no tiene ninguna).

En XQuery los outer joins se emulan realizando subconsultas FLOWR dentro de la cláusula return.

Veamos un ejemplo:

```
<html><head><title>Lista</title></head>
<body>
<h3>Numero de faltas por alumno</h3>
<table cols="2" width="50%" border="2px">
<tr><td>Alumno</td><td>Num Faltas</td></tr>
{
for $alum in doc("alumnos.xml")//alumno
order by $alum/nombre
return
  <tr><td> {string($alum/nombre)}</td>
    <td>
      {let $faltas := count(doc("faltas.xml")//falta[../@cial = $alum/@cial])
       return $faltas
      }
    </td>
  </tr>
}
</table>
</body>
</html>
```

Como puedes observar utilizamos la primera consulta FLOWR para generar las filas correspondientes a los alumnos. Dentro de la segunda celda de cada fila de la tabla utilizamos otra consulta FLOWR (en este caso con un let) para contar el número de faltas de el alumno seleccionado dentro de la primera consulta.

Como resultado aparecerán también los alumnos que no tenían faltas en el fichero faltas.xml:

Alumno	Num Faltas
Elena	0
Genaro	0
Irina	12
IvÁjn	3
Javier	5
Luis Fernando	0
Marcos	3
MarÁa	3
Miguel	12

Así, Elena aparece con cero faltas.



9.- Cuestiones finales.

9.1.- Ordenamiento.

Dentro de la cláusula ORDER BY se pueden especificar varios criterios de ordenación separados por comas (véase el caso de la práctica 17):

```
order by $alum/nombre, $falta/@fecha
```

Por cada uno de estos criterios podríamos especificar el orden ascendente o descendente. Las palabras reservadas son **ascending** y **descending**. El valor por defecto (cuando no especificamos nada) es ascending.

```
order by $alum/nombre ascending, $falta/@fecha descending
```

Sin embargo, el XML almacena datos conocidos como **semiestructurados**, es decir, puede haber nodos que no contengan toda la información. Imagina un alumno sin apellidos o un alumno sin notas. Para estos casos podemos especificar en la ordenación si los valores vacíos se consideran valores grandes (irán al final en una ordenación ascendente) o valores pequeños (irán al principio en una ordenación ascendente). Las palabras reservadas son **empty greatest** o **empty least**. La declaración con estas palabras reservadas afectará a los valores vacío y también a los valores NaN (Not a Number) que se obtienen como resultado de operaciones matemáticas defectuosas (por ejemplo hallar la media de un alumno al que le falta una nota). Veamos un ejemplo:

```
declare default order empty greatest;  
for $alum in doc("alumnos.xml")//alumno  
order by $alum/nombre ascending, $alum/apellidos ascending  
return $alum
```

En la primera línea especificamos que los valores vacíos son considerados como grandes (van al final), así como los valores NaN. Si hubiera un alumno sin nombre aparecería al final.

La última cuestión con el ordenamiento se produce cuando varios nodos tienen los mismos valores en el documento XML. En XML esto es mucho más frecuente que en las bases de datos relacionales. Para estos casos podemos indicar que en caso de valores iguales se respete el orden en que aparecen los nodos en el fichero XML. Esto se conoce como **ordenamiento estable**. Observa el ejemplo:

```
stable order by $alum/apellidos;
```

Como ves, basta con indicar la palabra reservada **stable** antes de la cláusula Order By.

9.2.- Comentarios.

Los comentarios en XQuery tienen la siguiente sintaxis:

```
(: esto es un comentario :)
```

9.3.- Comentario final.

El estándar XQuery puede llegar a imponerse como un estándar de uso cotidiano. Si bien será difícil que reemplace al SQL lo cierto es que la mayoría de las bases de datos están incorporando módulos que lo soportan. El principal motivo es el auge que la tecnología XML está consiguiendo en prácticamente todos los ámbitos de desarrollo.

Los apuntes contenidos en este tema son una aproximación al estándar, pero no lo hemos cubierto al 100%. Tiene, por ejemplo, construcciones IF, cuantificador EXIST, y se pueden hacer agrupamientos utilizando subconsultas.

No obstante sí que hemos visto una buena parte del estándar, que te permitirá obtener datos de ficheros XML muy diversos.



CICLO FORMATIVO DE GRADO SUPERIOR:

“DESARROLLO DE APLICACIONES WEB”



MÓDULO:

**Lenguajes de marcas y
sistemas de gestión de
información**

(4 horas semanales)