Approfondimenti di XML

Un utile editor per XML : http://pollo.sourceforge.net/

Ci sono varie ragioni per cui l'XML ha acquisito una certa importanza.

- L' XML crea documenti e dati indipendenti dall'applicazione (è esprtesso in un formato testo indipendente dall'applicazione).
- L'XML ha una sintassi standard per i metadati
- L'XML fornisce una struttura standard sia per i documenti che per i dati

Differenza fra documenti e dati: i documenti elettronici sono la controparte dei documenti cartacei. Sono una combinazione di contenuto e di presentazione. Il contenuto è costituito da proposizioni in linguaggio naturale che formano paragrafi e pagine. I dati sono campi (di qualche tipo es. interi, caratteri etc.) processabili dal computer. L' XML permette di aggiungere metadati nella forma di markup ad entrambi i tipi di dato.

In un documento XML il markup è separato dal contenuto e può contenere conteno.
Si veda il modulo su XML del corso di.....

Schemi XML

Uno scema è un linguaggio di definizione che permette di vincolare un documento XML ad un preciso vocabolario ed ad una precisa struttura. (Ha la stessa funzione delle DTD).
Quello che si vuole definire in uno schema XML sono
I tipi di elementi
I tipi degli attribuuti
I tipi complessi compisizione dei due tipi precedenti.

Tabella dei tipi di dato elementari definiti dall'XML schema

DATA TYPE	DESCRIPTION
string	Unicode characters of some specified length.
boolean	A binary state value of true or false.
ID	A unique identifier attribute type from the 1.0 XMl Specification.
IDREf	A reference to an ID.
integer	The set of whole numbers.
long	long is *derived* from integer by fixing the values of maxIndusive to be 9223372036854775807 and minIndusive to be -9223372036854775808.
int	int is *derived* from long by fixing the values of maxIndusive to be 2147483647 and minJndusive to be -2147483648.
short	short is derived from int by fixing the values of maxInclusive to be 32767 and minJndusive to be -32768.
decimal	Represents arbitrary precision decimal numbers with an integer part and a fraction part.
float	IEEE single precision 32-bit floating-point number.
double	IEEE double-precision 64-bit fioating-point number.
date	Date as a string defined in JSO 8601.
time	Time as a string defined in ISO 8601.

Abbiamo quindi uno schema che più generare varie istanze di documenti
Dei validatori possono usare lo schema per garantire la validità di un documento XML
Gli schemi sono essi stessi scritti in XML (a differenza delle DTD)

Un tipo semplice è un elemento costituito da un nome e da un vincolo di tipo sul valore.

Ad esempio la definizione dell' elemento autore che può contenere una stringa di un numero imprecisato di caratteri:

```
<xsd:element name=”author” type=”xsd:string” />
```

Una istanza dell'elemento autore potrebbe essere:

```
<author> Tim Berners-Lee </author>
```

Un elemento complesso può contenere sia attributi che altri elementi.

In questo esempio diciamo che l'elemento di nome “book” ha due attributi. Il primo di nome “title” è una stringa mentre il secondo di nome “pages” è un numero intero.

```
<xsd:element name="book">
   <xsd:complexType>
     <xsd:attribute name="title" type="xsd:string" />
     <xsd:attribute name="pages" type = "xsd:int" />
   </xsd:complexType>
</xsd:element>
```

Una istanza di questo schema potrebbe essere:

```
<book title = "More Java pitfalls" pages="453" />
```

Nel secondo esempio definiamo l'elememto product che contiene sia degli attribute che elementi figli. I tre attributi sono id (un identificatore unico) titolo (una stringa) e price (un numero con decimali).
Gli elementi figli sono “description” (una stringa) e categoria (una stringa).
I parametri `minOccurs e maxOccurs` indicano rispettivamente il minimo e il massimo numero di occorenze. Quidi l'elemento “description”è opzionale in quanto può apparire 0 o 1 volta, mentre l'elemento “category” è obbligatorio e ripetibile un numero qualsiasi di volte.

```
<xsd:element name="product">
<xsd:complexType>
   <xsd:sequence>
      <xsd:element name="description" type="xsd:string"
      minOccurs="O" maxOccurs = "1" />
      <xsd:element name="category" type="xsd:string"
      minOccurs = "1" maxOccurs = "unbounded" />
   </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID" />
    <xsd:attribute name="title" type="xsd:string" />
    <xsd:attribute name="price" type="xsd:decimal" />
  </xsd:complexType>
</xsd:element>
```

Una prima istanza di questo schema è:

```
<product id="PO1" title="Wonder Teddy" price="49.99">
   <description>
   The best selling teddy bear of the year.
   </description>
   <category> toys </category>
   <category> stuffed animals </category>
</product>
```

Una seconda istanza, con una struttura diversa, è:

```
<product id="PO2" title="RC Racer" price="89.99">
   <category> toys </category>
   <category> electronic </category>
   <category> radio-controlled </category>
</product>
```

Namespaces

I namespace sono meccanismi per creare nomi globali unici per gli elementi e gli attribute. Questo evita che ci siano conflitti fra linguaggi di markup differenti.

Ogni nome di XML è composto di due parti un “prefisso” ed una parte locale.
Ad esempio nello schema precedente

```
<xsd:string>
```

ha una parte lolcale “string” e un prefisso “xsd”. Il prefisso fa riferimento ad una definizione esterna il Namespace. Questo riferimento esterni è un URI (Uniform Resource Identificator).
Ad esempio il prefisso “xsd” negli schemi fa riferimento al Namespace:

```
http://www.w3.org/2001/XMLSchema
```

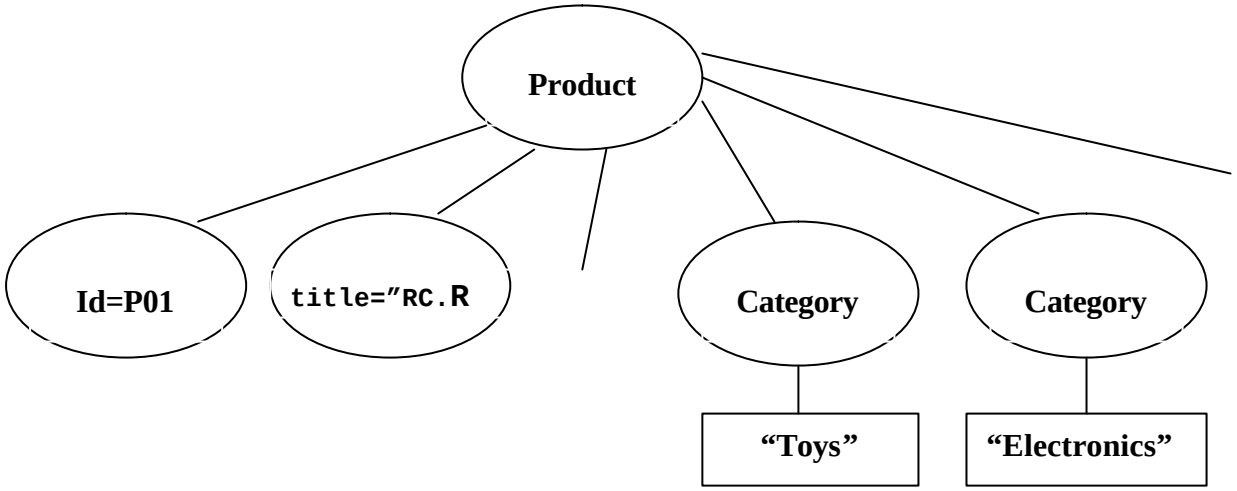
Una definizione per “xsd” potrebbe essere la seguente:

```
<xsd:schema xmlns:xsd=”http://www.w3.org/2001/XMLSchema”>
```

dove la definizione si applica all’ elemento “schema” (che conterrà tutto lo schema).
“xmlns: “ indica che stiamo definendo un namespace.

DOM – Document Object Model

Un documento XML puo essre visto come un albero. Ogni nodo rappresenta un elemento. Gli elementi interni ad un altro generano dei nodi figli. Anche gli attributi sono nodi figli dell'elemento in cui sono contenuti.



Parte dell' albero che rappresenta l'esempio precedente

Mentre il documento XML è memorizzato in un file su disco, DOM è una rappresentazione nello spazio di memoria di un programma. Questo programma può essere il browser, una applicazione Java etc.
Delle API (Application Programming Interfaces) permettono al programma di navigare nell'albero del documento

XPath

XPath è un linguaggio tramite il quale è possibile esprimere delle espressioni per indirizzare parti di un documento XML. È un linguaggio ideato per operare all'interno di altre tecnologie XML quali XSL ed altre di cui parleremo fra breve. Può essere utilizzato all'interno di URI o come valore di attributi di documenti XML. XPath opera su una rappresentazione logica del documento XML, che viene modellato con una struttura ad albero ed XPath definisce una sintassi per accedere ai nodi di tale albero. Oltre a questo XPath mette a disposizione una serie di funzioni per la manipolazione di stringhe, numeri e booleani, da utilizzare per operare sui valori o sugli attributi dei nodi.
Le espressioni definite da XPath per accedere ai nodi dell'albero prendono il nome di Location Path (percorsi di localizzazione).
La struttura un location path è la seguente: *axis::node-test[predicate]*.
La componente *axis* esprime la relazione di parentela tra il nodo cercato ed il nodo corrente; la componente *node-test* specifica il tipo o il nome del nodo da cercare; mentre *predicate* contiene zero o più filtri (espressi tra parentesi quadre) per specificare delle condizioni più selettive da applicare alla ricerca.

Le relazioni di parentela principali che possono essere contenute in *axis* sono:

- ancestor: indica tutti i nodi antenati del nodo corrente, ovvero tutti i nodi che lo precedono nell'albero associato al documento XML;
- attribute: indica tutti gli attributi del nodo corrente;
- child: indica i nodi figli del nodo corrente;
- descendant: indica tutti i discendenti del nodo corrente, ovvero tutti i nodi che hanno seguono il nodo corrente nell'albero XML;
- parent: indica il nodo genitore del nodo corrente, ovvero quello che lo precede nell'albero;
- self: indica il nodo corrente.

Vediamo qualche esempio per capire meglio come utilizzare i Location Path per accedere agli elementi di un documento XML, utilizzando l'esempio una rubrica telefonica.

```
<?xml version="1.0"?>
<rubrica>
	<persona>
		<nome>Mario</nome>
		<cognome>Rossi</cognome>
		<indirizzo>
			<via>via bianchi 1</via>
			<cap>00000</cap>
			<citta>Roma</citta>
		</indirizzo>
		<telefono>
			<telefono_fisso gestore="Abc">123456</telefono_fisso>
			<telefono_cellulare gestore="Def">987656412</telefono_cellulare>
		</telefono>
	</persona>
</rubrica>
```

child::nome
Questa espressione seleziona tutti i nodo chiamati 'nome' che sono figli del nodo corrente.
*child::**
Seleziona tutti i nodi figli del nodo corrente.
attribute::gestore
Seleziona l'attributo di nome 'gestore' del nodo corrente.
descendant::cognome[cognome='Rossi']
Seleziona tutti i nodi chiamati 'cognome' tra i nodi discendenti del nodo corrente, il cui valore è Rossi.

In XPath è possibile accedere ai nodi dell'albero utilizzando delle espressioni abbreviate dei Location Path. Le espressioni abbreviate, sono una versione semplificata e compatta dei Location Path ed offrono un meccanismo più veloce, ma al tempo stesso meno potente per accedere ai nodi dell'albero. Queste espressioni sono costituite da una lista di nomi di elementi del documento XML, separati da uno slash(/), e tale lista descrive il percorso per accedere all'elemento desiderato. È un meccanismo molto simile a quello usato per identificare i file e le directory nel filesystem. Ad esempio per indicare il file (chiamato mio-file) memorizzato all'interno di una directory (chiamata mia-directory) del vostro hard disk, utilizzate la seguente sintassi: c:\directory1\file1. Le espressioni abbreviate di XPath utilizzano un meccanismo concettualmente simile: vediamo come, utilizzando il solito file XML d'esempio.

/rubrica/persona/nome
Questa espressione abbreviata permette di recuperare i nodi chiamati 'nome' indicando il percorso assoluto per raggiungere il nodo desiderato, attraverso una lista di nodi separata da slash.
//nome
Con il doppio slash ricerchiamo i nodi chiamati 'nome', in tutto il documento, indipendentemente dalla loro posizione e dal loro livello sull'albero associato al documento XML.
/rubrica//via
Ricerchiamo tutti i nodi chiamati 'via' a qualsiasi livello dell'albero purchè contenuti all'interno del nodo chiamato 'rubrica'.
*/rubrica/persona/**
Ricerca qualsiasi elemento figlio del nodo chiamato 'persona'.
//telefono_fisso /@gestore
Ricerca l'attributo 'gestore' dell'elemento 'nome'.

Anche all'interno delle espressioni abbreviate possiamo inserire i predicati visti nel caso dei Location Path. Ad esempio:

//persona[nome='Mario']
questa espressione ricerca tutti i nodi 'persona' che hanno il tag 'nome' il cui valore è Mario.
Come detto all'inizio del capitolo, XPath mette a disposizione anche delle funzioni per gestire i nodi, le stringhe, i numeri e i booleani. Vediamo adesso di elencare brevemente e schematicamente alcune funzioni principali:

- count(node-set): restituisce il numero di nodi contenuti nell'insieme di nodi passato come argomento della funzione;
- name(nodo): restituisce il nome di un nodo;
- position(): determina la posizione di un elemento all'interno di un insieme di nodi;
- last(): indica la posizione dell'ultimo nodo di un'insieme di nodi;
- id(valore): seleziona gli elementi in funzione del loro identificatore;
- concat(s1,...,sn): restituisce una stringa risultato della concatenazione delle stringhe specificate tra gli argomenti di una funzione;

- string(valore): converte il valore dell'argomento in una stringa;
- string-length(stringa): ritorna la lunghezza della stringa passata come parametro;
- substring(stringa,inizio,lunghezza): restituisce una sotto-stringa della stringa passata come argomento;
- ceiling(numero): arrotonda il numero per eccesso;
- floor(numero): arrotonda il numero per difetto;
- number(valore): converte il valore dell'argomento in un numero;
- sum(node-set): esprime la somma di un insieme di valori numerici contenuti in un insieme di nodi.

Come esempi di espressioni XPath che fanno uso di queste funzioni consideriamo:

//persona[last()]
Questa espressione restituisce l'ultimo nodo 'persona' contenuto all'interno del file XML.
//persona[position()= 3]
Restituisce il terzo nodo 'persona' contenuto all'interno del file XML.
count(/rubrica/persona)
Indica il il numero di nodi chiamati 'persona' contenuti all'interno del documento XML.
La specifica di XPath, è all'indirizzo: http://www.w3.org/TR/xpath.

Style Sheet

Una caratteristica fondamentale di XML è quella di occuparsi esclusivamente della descrizione del contenuto dell'informazione e non della sua rappresentazione. Infatti i tag XML non esprimono in alcun modo come verrà visualizzato il loro contenuto, cosa che invece accade in altri linguaggi basati su marcatori, come HTML. L'informazione contenuta in un file XML può essere visualizzata definendo degli stili di rappresentazione, che, applicati al file XML, saranno in grado di rappresentarne il contenuto nel modo desiderato.
In XML questo può essere fatto utilizzando XSL (eXtensible Stylesheet Language). XSL è un linguaggio basato su XML per esprimere i fogli di stile, ossia un documento contenente le regole per rappresentare l'informazione. XSL è composto da due componenti: XSLT (eXtensible Stylesheet Language Transformation), che descriveremo in questo capitolo, e XSL-FO (eXtensible Stylesheet Language - Formatting Objects). All'interno di XSL viene utilizzato XPath per la ricerca ed il filtraggio dei contenuti del file XML.

XSLT

XSLT è un linguaggio basato su XML che permette di definire delle regole per trasformare un documento XML in un altro documento XML o in un documento HTML. Utilizzando XSL possiamo ad esempio visualizzare il contenuto di un documento XML in HTML, XHTML o SVG. La trasformazione viene realizzata da un XSLT Processor che riceve come input il file XML da trasformare, il file XSL con la definizione dello stylesheet da applicare e produce come output il file trasformato. Un file XSL è formato da una serie di template (modelli) che contengono le regole di trasformazione dei tag del documento XML. Questi template vengono applicati ai tag corrispondenti dal XSLT Processor in maniera ricorsiva nel corso della trasformazione.
Come esempio vediamo di definire un file XSL per visualizzare un file XML di esempio in una pagina HTML.
Il file XML di input è:

```
<?xml version="1.0"?>
<rubrica>
	<persona>
		<nome>Mario</nome>
		<cognome>Rossi</cognome>
		<indirizzo>
			<via>via bianchi 1</via>
			<cap>00000</cap>
			<citta>Roma</citta>
		</indirizzo>
		<telefono>
			<telefono_fisso>123456</telefono_fisso>
			<telefono_cellulare>987656412</telefono_cellulare>
		</telefono>
	</persona>
</rubrica>
```

Il file XSL con la definizione dello stile per la creazione del file HTML è:

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
	<xsl:template match="/">
		<html>
			<head>
				<title>Rubrica in versione HTML</title>
			</head>
			<body>
				<h1>Rubrica</h1>
				<xsl:apply-templates/>
			</body>
		</html>
	</xsl:template>
	<xsl:template match="persona">
	<h2> <xsl:value-of select="cognome"/> <xsl:value-of select="nome"/> </h2>
```

```
                <ul>
                        <li>Via: <xsl:value-of select="./indirizzo/via"/></li>
                        <li>CAP: <xsl:value-of select="./indirizzo/cap"/></li>
                        <li>Citta': <xsl:value-of select="./indirizzo/citta"/></li>
                        <li>Telefono (fisso): <xsl:value-of select="./telefono/telefono_fisso"/></li>
                        <li>Telefono (cellulare): <xsl:value-of select="./telefono/telefono_cellulare"/></li>
                </ul>
        </xsl:template>
</xsl:stylesheet>
```

In un file XSL le regole di trasformazione sono contenute all'interno degli elementi *template* e tramite l'attributo match possiamo specificare, utilizzando la sintassi XPath, il tag a cui si riferiscono queste regole. Nel nostro esempio il primo elemento *template* contiene le regole di trasformazione dell'elemento root del file di input (l'elemento <rubrica>); mentre il secondo definisce le regole per la trasformazione degli elementi <persona>. Il processore XSLT effettua il parsing del documento XML da trasformare e, per ogni nodo incontrato, ricerca il *template* appropriato all'interno del file XSL. Quando il processore incontra il nodo root del documento XML applica il primo *template* e quindi costruisce lo scheletro del file HTML. Con l'elemento <xsl:apply-templates> si indica al processore XSLT di analizzare i nodi figli del nodo corrente alla ricerca di altri *template* da applicare.

Il browser applica direttamente la trasformazione XSL al file XML e il risultato e' mostrato in figura

Il prodotto dalla trasformazione XSLT è il seguente codice HTML:

```
<html>
	<head>
		<title>Rubrica in versione HTML</title>
	</head>
	<body>
		<h1>Rubrica</h1>
		<h2>Rossi Mario</h2>
		<ul>
			<li>Via: via bianchi 1</li>
			<li>CAP: 00000</li>
			<li>Citta': Roma</li>
			<li>Telefono (fisso): 123456</li>
			<li>Telefono (cellulare): 987656412</li>
		</ul>
	</body>
</html>
```

L’esempio riguardava la trasformazione da XML a HTML. Ma sono possibili altre forme di trasformazione. Si possono produrre file PDF o RTF o fare qualsiasi altra cosa.

XQuery

In questo capitolo andiamo ad analizzare una tecnologia ideata per il recupero delle informazioni memorizzate all'interno di un file XML. È molto importante per la diffusione e l'utilizzo di XML nell'ambito di documenti contenenti grandi quantità di dati, avere a disposizione uno strumento relativamente facile e potente per poter recuperare l'informazione presente in un file XML.

Questo strumento deve permettere di realizzare delle query (interrogazioni) sul documento proprio come avviene ad esempio con il linguaggio SQL nel caso dei database relazionali.

XML Query languge (XQuery) nasce proprio con l'intento di realizzare un linguaggio per recuperare agevolmente le informazioni da un documento XML ed andare a costituire una sorta di "SQL per XML". XQuery non è un linguaggio basato su XML ed è costituito da una sintassi semplice e facilmente leggibile per formulare, nel modo più agevole possibile, le query sui dati. Il working group del W3C ha sviluppato anche ua versione di XQuery con sintassi XML, chiamata XQueryX di cui parleremo in seguito.

Prima di iniziare a parlare in dettaglio di XQuery, introduciamo il documento XML di esempio che utilizzeremo per realizzare le query nel corso degli esempi.

Consideriamo l'esempio di un documento XML (chiamato arch_libri.xml) nel quale sono memorizzate le informazioni relative ad un archivio formato da numerosi libri dove, per ogni libro, l'informazione viene strutturata nel seguente modo (per semplicità ne riportiamo solamente la struttura base):

```
<libro>
        <titolo>Titolo_del_libro</titolo>
        <autore>Autore_del_libro</autore>
        <editore>Editore_del_libro</editore>
        <prezzo>Prezzo_del_libro</prezzo>
</libro>
```

Una query in XQuery è costituita da un'espressione che legge una sequenza di nodi XML od un singolo valore e restituisce come risultato una sequenza di nodi od un singolo valore. Le espressioni XQuery sono composte da espressioni XPath per individuare i nodi da analizzare e da delle funzionalità aggiuntive specifiche di XQuery per il recupero delle informazioni.

Ad esempio:

```
document("arch_libri.xml")//libro[prezzo > 50] SORTBY (autore)
```

Questa query ricerca all'interno del nostro documento d'esempio tutti i nodi <libro> che hanno un prezzo maggiore di 50 e ordina il risultato in funzione del nome dell'autore.

L'espressione principale utilizzata in XQuery, per formulare interrogazioni complesse, è del tipo **For-Let-Where-Return**. Questa espressione costituisce una generalizzazione del costrutto SELECT-FROM-HAVING-WHERE del linguaggio SQL e se avete già dimestichezza con le query SQL sarete sicuramente avvantaggiati nella comprensione degli esempi di espressioni XQuery.

Consideriamo il seguente esempio:

```
FOR $e IN document("arch_libri.xml")//editore
LET $l := document("arch_libri.xml")//libro[editore=$e]
WHERE count($l) > 5
RETURN
<risultato>
        {$e}
</risultato>
```

La prima istruzione crea una lista (associata alla variabile '$p') contenente tutti gli editori presenti nel nostro archivio. La seconda riga associa a ciascun editore, la lista dei libri da lui editi (variabile '$l'), andando a creare una lista ordinata di tuple formate da ($e,$l). Tramite la terza riga, determiniamo un filtro sulla risposta andando a considerare solamente gli editori hanno pubblicato più di cinque libri. L'ultima istruzione (RETURN) crea il risultato inserendo all'interno di un elemento chiamato <risultato>, i nodi <editore> che soddisfano i criteri della nostra query.

Un secondo esempio è questo: la seguente espressione XQuery che permette di ricercare i titoli di tutti i libri scritti da 'Mario Rossi':

```
FOR $l IN document("arch_libri.xml")//libro
WHERE $l/autore="Mario Rossi"
RETURN
        <risultato>
                $l/titolo
        </risultato>
```

La funzione **count()** presente in questo esempio è una delle funzioni che XQuery matte a disposizione per operare sulle liste di elementi. Le funzioni principali che XQuery offre oltre a count() (che restituisce il numero di elementi presenti) sono:

- **avg()**, per calcolare il valor medio dei valori degli elementi
- **union()**, **intersection()**, **difference()** che realizzano operazioni 'insiemistiche' sugli elementi.

Costrutti particolari

XQuery supporta anche il costrutto IF-THEN-ELSE all'interno delle sue espressioni.
Ad esempio:

```
FOR $l IN document("arch_libri.xml")//libro
RETURN
        <risultato>
                {
                        IF ($l/editore ='Editore1')
                                THEN $l/titolo
                                ELSE $l/autore
                }
        </risultato>
```

Il risultato di questa query contiene i titoli dei libri se l'editore è 'Editore1', altrimenti il nome degli autori dei libri delle altre case editrici.

Altri due costrutti molo utili in XQuery sono SOME-IN-SATISFIES e EVERY-IN-SATISFIES, che permettono di verificare determinate proprietà per gli elementi contenuti in una lista.

Ad esempio:

```
FOR $l IN document("arch_libri.xml")//libro
WHERE SOME $t IN $l/titolo SATISFIES (contains($t,"XML") AND contains($t,"tutorial"))
RETURN
        <risultato>
                { $l/titolo }
        </risultato>
```

```
FOR $l IN document("arch_libri.xml")//libro
WHERE EVERY $t IN $l/titolo SATISFIES contains($t,"XML")
RETURN
        <risultato>
                { $l/titolo }
        </risultato>
```

La prima query restituisce come risultato il titolo di tutti i libri nei cui titoli compare contemporaneamente sia la stringa 'XML' che la stringa 'tutorial'; mentre la seconda query restituisce i titoli dei libri che contengono la stringa 'XML' nel loro titolo.

Le specifiche di Xquery sono reperibili all'indirizzo http://www.w3.org/TR/xquery. Un altro linguaggio collegato è Xupdate che permette di modificare i documenti XML.

Database XML

I database XML permettono di memorizzare documenti XML in modo “nativo”. Si può cercare l’informazione usando Xquery o modificarla con Xupdate.

Un esempio di database XML open source è eXist. http://exist-db.org/index.html

Una demo dove potete provare Xquery è http://demo.exist-db.org/exist/xquery/xquery.xq