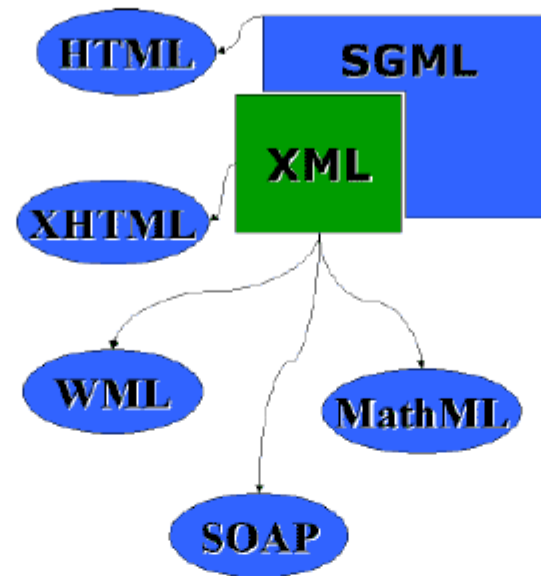


Basi di dati per XML
Prof. Alfredo Pulvirenti
Prof. Salvatore Alaimo

- XML (***eXtensible Markup Language***) è un meta linguaggio.
- Può essere definito come un insieme di regole e convenzioni che consentono di descrivere qualunque linguaggio di markup.
- Esso è quindi basato su **marcatori** che possono essere definiti in base alle proprie esigenze.

- L'idea è in parte derivata dai concetti di base di SGML (Standard Generalized Markup Language).

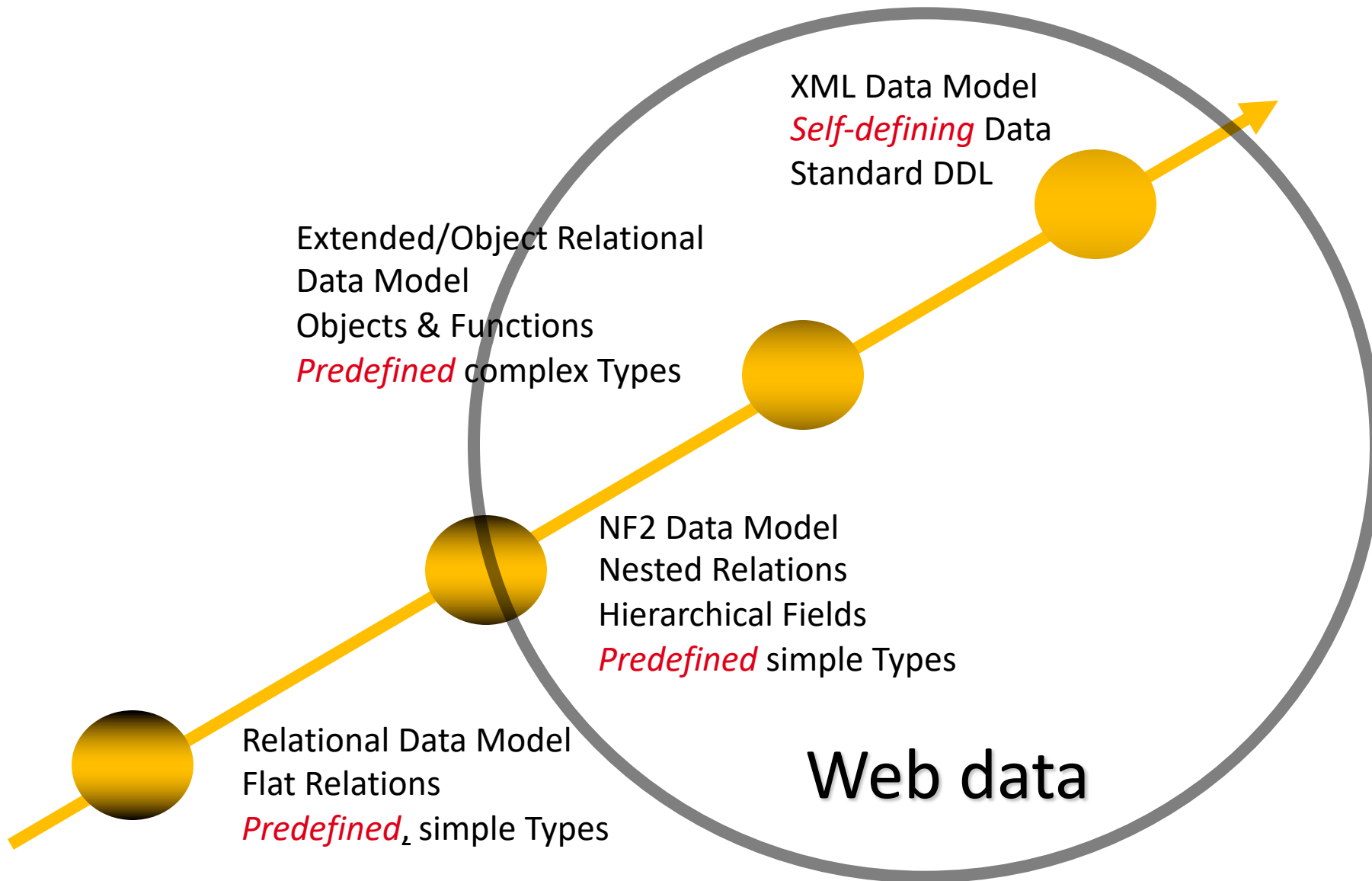


- XML è un linguaggio a marcatori estendibile.
- I linguaggi a marcatori consentono di descrivere con precisione qualsiasi tipo di informazione: gerarchica, lineare, relazionale o binaria.
- In XML l'informazione viene organizzata utilizzando una struttura gerarchica che è possibile scorrere e navigare con semplicità e al suo interno è possibile ricercare le informazioni desiderate.

- XML non definisce la propria collezione di marcatori (a differenza di HTML) ma definisce le regole sintattiche attraverso le quali è possibile generare dei marcatori personalizzati e i loro eventuali attributi.

- La sintassi di XML è formata essenzialmente da tag i quali possono avere attributi ed eventualmente al proprio interno altri tag.
- I tag devono essere a coppie, ci deve essere la presenza contemporanea dei tag di apertura e chiusura.
- La presenza dei tag è necessaria (così come in HTML) per dividere il contenuto informativo del documento dalla sintassi utilizzata per rappresentarlo.

Evoluzione dei modelli dei dati



- La struttura di un documento XML è gerarchica e ad albero.
- Es.

```
<?xml version="1.0"?>  
<tag1>  
  <tag2> contenuto informativo </tag2>  
</tag1>
```


- HTML ed XML hanno una relazione molto stretta. Infatti è possibile scrivere un documento HTML in XML.

```
<?xml version="1.0"?>
<html>
  <head>
    <title> Titolo del documento HTML</title>
  </head>
  <body>
    Un documento HTML scritto in XML
  </body>
</html>
```

Diamo al documento appena creato estensione xml

Regole di base

```
<?xml version="1.0"?>
<Gran_Premio>
  <Nazione>Belgio</Nazione>
  <Circuito>SPA</Circuito>
  <Anno>2001</Anno>
  <Griglia_Di_Partenza>
    <Pilota>
      <Posizione>1</Posizione>
      <Nome>Juan Pablo Montoya</Nome>
      <Vettura>Williams BMW</Vettura>
      <Tempo>1.52.072</Tempo>
    </Pilota>
    <Pilota>
      <Posizione>2</Posizione>
      <Nome>Ralph Shumacher</Nome>
      <Vettura>Williams BMW</Vettura>
      <Tempo>1.53.279</Tempo>
    </Pilota>
    <Pilota>
      ...
    </Pilota>
  </Griglia_Di_Partenza>
</Gran_Premio>
```

Identifica il tipo di documento e
Specifica la versione di XML utilizzata

Tag di apertura e chiusura

Regole di base

```
<?xml version="1.0"?>
<Gran_Premio>
  <Nazione>Belgio</Nazione>
  <Circuito>SPA</Circuito>
  <Anno>2001</Anno>
  <Griglia_Di_Partenza>
    <Pilota>
      <Posizione>1</Posizione>
      <Nome>Juan Pablo Montoya</Nome>
      <Vettura>Williams BMW</Vettura>
      <Tempo>1.52.072</Tempo>
    </Pilota>
    <Pilota>
      <Posizione>2</Posizione>
      <Nome>Ralph Shumacher</Nome>
      <Vettura>Williams BMW</Vettura>
      <Tempo>1.53.279</Tempo>
    </Pilota>
    <Pilota>
      ...
    </Pilota>
  </Griglia_Di_Partenza>
</Gran_Premio>
```

Identifica il tipo di documento e
Specifica la versione di XML utilizzata

Tag di apertura e chiusura

Elementi

Regole di base

```
<?xml version="1.0"?>
<Gran_Premio>
  <Nazione>Belgio</Nazione>
  <Circuito>SPA</Circuito>
  <Anno>2001</Anno>
  <Griglia_Di_Partenza>
    <Pilota>
      <Posizione>1</Posizione>
      <Nome>Juan Pablo Montoya</Nome>
      <Vettura>Williams BMW</Vettura>
      <Tempo>1.52.072</Tempo>
    </Pilota>
    <Pilota>
      <Posizione>2</Posizione>
      <Nome>Ralph Shumacher</Nome>
      <Vettura>Williams BMW</Vettura>
      <Tempo>1.53.279</Tempo>
    </Pilota>
    <Pilota>
      ...
    </Pilota>
  </Griglia_Di_Partenza>
</Gran_Premio>
```

Identifica il tipo di documento e
Specifica la versione di XML utilizzata

Tag di apertura e chiusura

Elementi

Il tag radice contiene tutti gli altri.
Questo viene chiamato “root element”

Componenti di un doc XML

- Prologo
 - È costituito da tutta la parte del documento XML che precede l'elemento root.
 - Gli attributi sono:
 - **version:** (obbligatorio) la versione di XML usata.
 - **encoding:** (opzionale) nome della codifica dei caratteri usata nel documento. (default: UTF-8 o 16)
 - **standalone:** (opzionale) se vale yes indica che il file non fa riferimento ad altri file esterni. (default: no)

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

<!-- questo e' un commento, questo documento XML descrive la gerarchia del corso Web II e
la descrizione dei temi trattati -->

<Corso
docente="Alfredo Pulvirenti"
nome_corso="TEC-23"
locazione="INGV Catania">
<Introduzione>...</Introduzione>
<Temi>
  <Tema
    numero="1"
    titolo="Uno sguardo al Web">
    ...
  </Tema>
  <Tema
    numero="2"
    titolo="Il linguaggio HTML">
    ...
  </Tema>
  <Tema
    numero="3"
    titolo="Linguaggi per il web server side: il PHP">
    ...
  </Tema>
  <Tema
    numero="4"
    titolo="XML">
    ...
  </Tema>
  <Tema
    numero="5"
    titolo="XHTML">
    ...
  </Tema>
  <Tema
    numero="6"
    titolo="Linguaggi per il web client side: Javascript">
    ...
  </Tema>
</Temi>
</Corso>
```

Componenti di un doc XML

- Elemento radice

```
<?xml
version="1.0"
encoding="UTF-8"
standalone="yes"?>
<!-- questo e' un commento, questo documento XML descrive la gerarchia del corso Web II e
la descrizione dei temi trattati -->
<Corso
docente="Alfredo Pulvirenti"
nome_corso="TEC-23"
locazione="INGV Catania">
<Introduzione>...</Introduzione>
<Temi>
  <Tema
    numero="1"
    titolo="Uno sguardo al Web">
    ...
  </Tema>
  <Tema
    numero="2"
    titolo="Il linguaggio HTML">
    ...
  </Tema>
  <Tema
    numero="3"
    titolo="Linguaggi per il web server side: il PHP">
    ...
  </Tema>
  <Tema
    numero="4"
    titolo="XML">
    ...
  </Tema>
  <Tema
    numero="5"
    titolo="XHTML">
    ...
  </Tema>
  <Tema
    numero="6"
    titolo="Linguaggi per il web client side: Javascript">
    ...
  </Tema>
</Temi>
</Corso>
```

Componenti di un doc XML

```
<?xml-stylesheet  
type="text/css"  
href="esempio.css"?>
```

```
<!DOCTYPE Corso SYSTEM "corso.dtd">
```

- Questa istruzione può essere presente nel prologo dopo la dichiarazione XML, associa un foglio di stile al documento xml.
- Ad un documento XML possono essere associate regole grammaticali. Queste ne descrivono gli aspetti semantici e consentono l'eventuale validazione automatica.

Regole fondamentali

- I nomi degli elementi sono **case-sensitive**.
- Ogni elemento aperto deve essere chiuso entro la fine del documento.
- Gli elementi possono essere nidificati, e in tal caso vanno chiusi esattamente nell'ordine inverso a quello di apertura.
- Un documento XML deve avere un unico elemento “radice”, in cui tutti gli altri sono nidificati

- Il **tag di apertura** di un elemento ha la forma seguente:

<nome attributi>

- **nome** è il nome dell' elemento.
- **attributi** è una lista di attributi per l' elemento (che può non apparire).
- Il **tag di chiusura** corrispondente ha la forma seguente:

</nome>

Notazione abbreviata dei tag senza valori

<nome/>

Attributi

Gli attributi permettono di specificare proprietà degli elementi come coppie nome-valore.

Sono usati per definire proprietà che non possono o non si vogliono inserire nel contenuto dell' elemento.

Vengono specificati all'interno dei tag di apertura degli elementi.

Al contrario degli elementi, per gli attributi l' ordine di presentazione non è significativo.

```
<?xml version="1.0"?>
<Gran_Premio>
  <Nazione>Belgio</Nazione>
  <Circuito>SPA</Circuito>
  <Anno>2001</Anno>
  <Griglia_Di_Partenza>
    <Pilota Posizione="1" Tempo="1.52.072">
      <Nome>Juan Pablo Montoya</Nome>
      <Vettura>Williams BMW</Vettura>
    </Pilota>
    <Pilota Posizione="2" Tempo="1.53.279">
      <Nome>Ralph Shumacher</Nome>
      <Vettura>Williams BMW</Vettura>
    </Pilota>
  </Griglia_Di_Partenza>
</Gran_Premio>
```

Regole Generali

- I nomi degli attributi sono **case-sensitive**.
- Lo stesso tag non può contenere due attributi con lo stesso nome.
- Non sono ammessi attributi senza valore (solo nome).
- Il valore degli attributi deve essere specificato **tra virgolette semplici o doppie**.
- Il valore può contenere **riferimenti ad entità**.
- Il valore non può contenere markup, sezioni CDATA o virgolette uguali a quelle iniziali.

<nome-elemento attributo="valore">

- Una **lista di attributi** si ottiene elencando più attributi separati da uno o più spazi:

<nome-elemento att1="vl1" att2="vl2">

- Per includere **virgolette** nel valore, è necessario usare un tipo diverso da quello usato per delimitare il valore stesso:

<nome-elemento att1= ' "virgolette" ' >

- Si possono includere **riferimenti a entità** nel valore:

<nome-elemento att1="" salve "">

- Alcune entità sono predefinite nel linguaggio XML e permettono di inserire quei caratteri che altrimenti sarebbero inutilizzabili. Le entità predefinite sono le seguenti:

entità	Significato
& ;	&
< ;	<
> ;	>
' ;	'
" ;	"

I namespace

- Spesso accade che alcuni nomi di elementi o attributi usati all'interno di un documento XML entrino in conflitto. Ad esempio il nome dell'elemento titolo, potrebbe indicare il titolo di un libro o di un dipinto.
- XML fornisce un utile meccanismo in grado di definire degli spazi di nomi (chiamati *namespace*) per risolvere queste ambiguità.
- Namespace o Dizionari
- Un namespace consiste di un gruppo di elementi e di nomi di attributi.
- I nomi del namespace vengono identificati utilizzando la seguente sintassi:
`ns-prefix:local-name`

- Ad esempio potremmo distinguere i tag come `<libro:titolo>` e `<corso:titolo>`.
- Un namespace deve essere dichiarato attraverso l'attributo **xmlns** prima di poterlo utilizzare all'interno di elemento.
- Ad esempio possiamo definire il namespace *libro* nel seguente modo:

```
<biblioteca xmlns:libro="http://www.esempio.org/1999/libro">  
  <libro:titolo>...</libro:titolo>  
</biblioteca>
```

- L'attributo xmlns definisce il namespace libro identificandolo univocamente con un *URI (Uniform Resource Identifier)* che nel nostro esempio è `http://www.esempio.org/1999/libro`.

<?xml version="1.0"?>

<clienti>

<cliente>

<ragione_sociale>Synapses srl</ragione_sociale>

<partita_iva>1234566779</partita_iva>

<indirizzo>Pezza Grande, Zona Industriale, CT</indirizzo>

<citta>Catania</citta>

<telefono>44445555</telefono>

<indirizzo><http://www.synapseslab.com></indirizzo>

<email>info@synapseslab.com</email>

</cliente>

<cliente>

...

</cliente>

</clienti>

- Il tag radice del documento XML (nel nostro caso <clienti>) conterrà l' indicazione del dizionario.
- Supponiamo che l' azienda abbia due punti vendita dove collezione dati relativi a clienti.
 - Catania
 - Ragusa

<?xml version="1.0"?>

<clienti

xmlns:ct="http://catania.aligroup.it/Dizionario/1.0"

xmlns:rg="http://ragusa.aligroup.it/Dizionario/1.0"

xmlns="http://aligroup.it/Dizionario/1.0"

>

<?xml version="1.0"?>

<clienti

xmlns:ct="http://catania.aligroup.it/Dizionario/1.0"

xmlns:rg="http://ragusa.aligroup.it/Dizionario/1.0"

xmlns="http://aligroup.it/Dizionario/1.0">

<cliente>

<ragione_sociale>Trinacria S.P.A.</ragione_sociale>

<partita_iva>1234566779</partita_iva>

<ct:indirizzo>Pezza Grande, Zona Industriale,CT</ct:indirizzo>

<citta>Catania</citta>

<telefono>44445555</telefono>

<rg:indirizzo>http://www.trinacria.it</rg:indirizzo>

<email>segreteria@trinacria.it</email>

</cliente>

<cliente>...</cliente>

</clienti>

<radice

xmlns:prefisso="URI"

xmlns="URI"

>

- URI viene utilizzato per la definizione UNIVOCHE del dizionario (namespace).
- Ha la forma di un indirizzo web, ma l'obiettivo è quello di definire l'univocità del namespace.
- Non ha nessuna relazione con il reale indirizzo web.

Riferimenti

- www.w3schools.com
- www.w3schools.com/xml
- www.w3schools.com/dtd
- <http://www.w3schools.com/schema/>

- Nasce l' esigenza di individuare all' interno di documenti XML elementi su cui operare.
- XPATH è linguaggio di XML per trovare informazioni dentro un documento XML.
- XPATH viene usato per “navigare” tra gli elementi e gli attributi di un documento XML.
- E' alla base di XSLT, XQuery e XPointer

Path expression in XPath

- Idea: usare una sintassi simile a quella dei pathname dei file per “navigare” la struttura ad albero di un documento
- Una espressione XPath è una stringa contenente nomi di elementi e operatori di navigazione e selezione:
 - . Nodo corrente
 - .. Nodo padre del nodo corrente
 - / nodo radice, o figlio del nodo corrente
 - // discendente del nodo corrente
 - @ attributo del nodo corrente
 - * qualsiasi nodo
 - [p] predicato (se l'espressione p, valutata, ha valore booleano)
 - [n] posizione (se l'espressione n, valutata, ha valore numerico)

Esempi base di path expressions

- Una path expression può iniziare con

doc(posizione_documento)

Restituisce l'elemento radice del documento specificato e tutto il suo contenuto: *doc("libri.xml")*

- A partire dalla radice del documento si possono specificare delle espressioni per estrarre il contenuto desiderato

- Esempio:


doc("libri.xml")/Elenco/Libro

Restituisce la sequenza di tutti gli elementi di tipo *Libro* contenuti nel documento *libri.xml*

Esempi di path expressions

```
<?xml version="1.0"?>
<Elenco>
  <Libro disponibilità='S'>
    <Titolo>Il Signore degli Anelli</Titolo>
    <Autore>J.R.R. Tolkien</Autore>
    <Data>2002</Data>
    <ISBN>88-452-9005-0</ISBN>
    <Editore>Bompiani</Editore>
  </Libro>
  <Libro disponibilità='N'>
    <Titolo>Il nome della rosa</Titolo>
    <Autore>Umberto Eco</Autore>
    <Data>1987</Data>
    <ISBN>55-344-2345-1</ISBN>
    <Editore>Bompiani</Editore>
  </Libro>
  <Libro disponibilità='S'>
    <Titolo>Il sospetto</Titolo>
    <Autore>F. Dürrenmatt</Autore>
    <Data>1990</Data>
    <ISBN>88-07-81133-2</ISBN>
    <Editore>Feltrinelli</Editore>
  </Libro>
</Elenco>
```

doc ("libri.xml")/Elenco/Libro



```
<Libro disponibilità='S'>
  <Titolo>Il Signore degli Anelli</Titolo>
  <Autore>J.R.R. Tolkien</Autore>
  <Data>2002</Data>
  <ISBN>88-452-9005-0</ISBN>
  <Editore>Bompiani</Editore>
</Libro>
<Libro disponibilità='N'>
  <Titolo>Il nome della rosa</Titolo>
  <Autore>Umberto Eco</Autore>
  <Data>1987</Data>
  <ISBN>55-344-2345-1</ISBN>
  <Editore>Bompiani</Editore>
</Libro>
<Libro disponibilità='S'>
  <Titolo>Il sospetto</Titolo>
  <Autore>F. Dürrenmatt</Autore>
  <Data>1990</Data>
  <ISBN>88-07-81133-2</ISBN>
  <Editore>Feltrinelli</Editore>
</Libro>
```


Condizioni su elementi/attributi

- Esempio:

`doc ("libri.xml")/Elenco/Libro[Editore='Bompiani']/Titolo`

Restituisce la sequenza di tutti i titoli dei libri dell'editore Bompiani che si trovano nel documento

Risultato:

```
<Titolo>Il Signore degli Anelli</Titolo>  
<Titolo>Il nome della rosa</Titolo>
```

Ricerca di sotto-elementi a qualsiasi livello

- Esempio:

`doc("libri.xml")//Autore`

Restituisce la sequenza di tutti gli autori che si trovano nel documento libri.xml, annidati a qualunque livello

```
<Autore>J.R.R. Tolkien</Autore>  
<Autore>Umberto Eco</Autore>  
<Autore>F. Dürrenmatt</Autore>
```

Condizione sulla posizione dei sottoelementi e uso di wildcard

- Esempio:

`doc ("libri.xml")/Elenco/Libro[2]/*`

Restituisce tutti i sottoelementi (*) contenuti nel **secondo** libro del documento `libri.xml`

```
<Titolo>Il nome della rosa</Titolo>  
<Autore>Umberto Eco</Autore>  
<Data>1987</Data>  
<ISBN>55-344-2345-1</ISBN>  
<Editore>Bompiani</Editore>
```

XQuery

- Linguaggio “alla SQL” per l’interrogazione di dati XML, definito da W3C
- Si basa su XPath per identificare frammenti XML

**È BASATO SULLA ELABORAZIONE
DI SEQUENZE DI NODI**

Espressioni FLWOR

- Una interrogazione XQuery è un'espressione complessa che consente di **estrarre parti di un documento e costruire un altro documento**
- Si basa (tipicamente) su 5 clausole (cfr SQL):
 - **FOR** iterare i valori di variabili su sequenze di nodi
 - **LET** legare variabili a intere sequenze di nodi
 - **WHERE** esprimere condizioni sui legami effettuati
 - **ORDER BY** imporre un ordinamento alla sequenza risultante
 - **RETURN** costruire il risultato (strutturato)

Espressioni FOR

- Esempio:

```
for $libro in doc("libri.xml")//Libro  
return $libro
```

- La clausola **for** valuta la path expression, che restituisce una sequenza di elementi, e la variabile **\$libro** **itera** all'interno della sequenza, assumendo ad ogni iterazione il valore di un nodo (libro) diverso
- La clausola **return** costruisce il risultato, in questo caso l'interrogazione restituisce semplicemente ogni valore legato a **\$libro** - cioè di tutti i libri del documento

Espressioni FOR annidate

- Le espressioni FOR possono essere annidate:

```
for $libro in doc("libri.xml")//Libro
```

```
    for $autore in $libro/Autore
```

```
    return $autore
```

- Semantica: per ogni valore di \$libro (libro), per ogni valore di \$autore (un autore *del libro corrente*), inserisci nel risultato l'autore legato a \$autore

Espressioni LET

- Consentono di introdurre nuove variabili:

```
let $libri := doc("libri.xml")//Libro  
return $libri
```

- La clausola **let** valuta l'espressione (//Libro) e assegna alla variabile \$libri l'intera sequenza restituita
- La valutazione di una clausola **let** assegna alla variabile *un singolo valore*: l'intera sequenza dei nodi che soddisfano l'espressione
- La query dell'esempio precedente è esprimibile come:

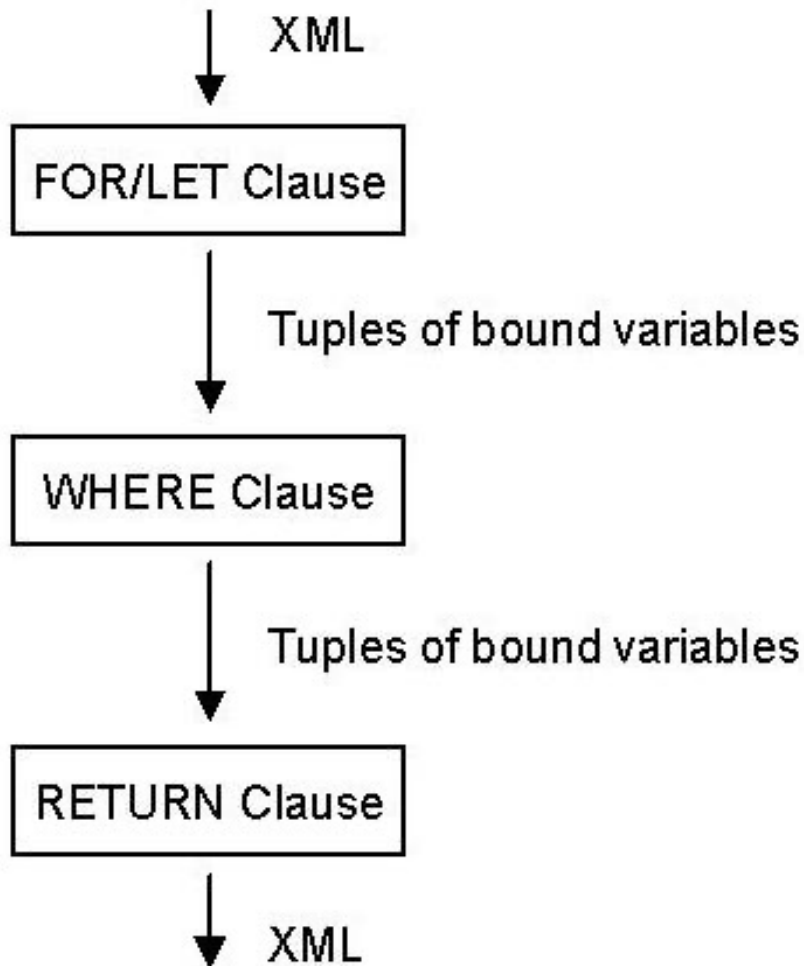
```
for $libro in doc("libri.xml")//Libro  
let $a := $libro/Autore   (: $a vale l'intera sequenza degli autori del libro :)  
return $a
```


Espressioni LET

```
for $libro in doc("libri.xml")//Libro
let $a := $libro/Autore
return <libro>
    <titolo> { $libro/titolo } </titolo>
    <autori> { $a } </autori>
</libro>
```

```
for $libro in doc("libri.xml")//Libro
for $a in $libro/Autore
return <libro>
    <titolo> { $libro/titolo } </titolo>
    <autore> { $a } </autore>
</libro>
```

FLWR expressions: interpretazione



- FOR : iterazione
 - ogni valore nella sequenza partecipa a una diversa “tupla di legami”
- LET : assegnamento
 - di una sequenza a una variabile (non aumenta il numero di tuple di legami)
- WHERE : filtraggio
 - viene valutata su ogni tupla separatamente, filtrandole in base alle condizioni espresse
- RETURN : ricostruzione
 - è eseguita una volta per ciascuna tupla di legami

Clausola WHERE

- La clausola WHERE esprime una condizione: solamente le tuple che soddisfano tale condizione vengono utilizzate per invocare la clausola RETURN
- Le condizioni nella clausola WHERE possono contenere diversi predicati connessi da AND e OR. Il **not()** è realizzato tramite una funzione che inverte il valore di verità
- Esempio:

```
for $libro in doc ("libri.xml")//Libro
where $libro/Editore="Bompiani"
      and $libro/@disponibilità="S"
return $libro
```
- Restituisce tutti i libri pubblicati da Bompiani che sono disponibili

Clausola WHERE

- Spesso le clausole where possono essere omesse usando opportune Path Expression

- Esempio:

```
for $libro in  
  doc("libri.xml")//Libro[Editore="Bompiani" and  
                           @disponibilità="S"]  
return $libro
```

- Restituisce tutti i libri pubblicati da Bompiani che sono disponibili

Clausola RETURN

- Genera l'output di un'espressione FLWR che può essere:

- Un nodo
- Una “foresta” ordinata di nodi
- Un valore testuale (PCDATA)

<Autore>F. Dürrenmatt</Autore>

<Autore>J.R.R. Tolkien</Autore>
<Autore>Umberto Eco</Autore>
<Autore>F. Dürrenmatt</Autore>

F. Dürrenmatt

- Può contenere dei **costruttori di nodi**, dei valori *costanti*, riferimenti a *variabili* definite nelle parti FOR e LET, ulteriori *espressioni annidate*

Clausola RETURN

- Un *costruttore di elemento* consta di un tag iniziale e di un tag finale che racchiudono una lista (opzionale) di espressioni annidate che ne definiscono il contenuto
- Esempio:

```
for $libro in doc("libri.xml")//Libro  
where $libro/Editore="Bompiani"  
return <LibroBompiani>  
      { $libro/Titolo }  
      </LibroBompiani>
```

nuovo
elemento

espressione
annidata

```
<LibroBompiani><Titolo>Il Signore degli Anelli</Titolo></LibroBompiani>  
<LibroBompiani><Titolo>Il nome della rosa</Titolo></LibroBompiani>
```

Clausola RETURN

- Esempio (variante):

```
for $libro in doc("libri.xml")//Libro
where $libro/Editore="Bompiani"
return <Libro-Bompiani>
      { $libro/Titolo/text() }
</Libro-Bompiani>
```

estrae il solo
contenuto PCDATA
di un elemento

```
<Libro-Bompiani>Il Signore degli Anelli</Libro-Bompiani>
<Libro-Bompiani>Il nome della rosa</Libro-Bompiani>
```

“Cardinalità” delle sequenze costruite nel risultato

- La clausola return è eseguita tante volte quanti sono i distinti assegnamenti delle “tuples of bound variables”
- Nel seguente esempio, in base alla semantica della clausola let, la return è eseguita una sola volta
 - (si nota che il nuovo tag <Libro-Bompiani> è creato una sola volta):

```
let $libri := doc("libri.xml")//Libro[Editore="Bompiani"]
return <Libro-Bompiani>
      { $libri/Titolo }
      </Libro-Bompiani>
```

```
<LibroBompiani>
  <Titolo> Il Signore degli Anelli </Titolo>
  <Titolo> Il nome della rosa </Titolo>
</LibroBompiani>
```

Path expression che inizia da una sequenza

Ordinare il risultato

- Esempio:

```
for $libro in doc("libri.xml")//Libro
order by $libro/Titolo
return
    <Libro>
        { $libro/Titolo,
          $libro/Editore }
    </Libro>
```

- I libri vengono ordinati rispetto al titolo
 - I matching della variabile, inizialmente generati in “document order”, sono riordinati prima di essere passati alla clausola return per generare il risultato

Funzioni aggregate

- Esempio:

```
for $e in doc("libri.xml")//Editore
let $libro := doc("libri.xml")//Libro[Editore = $e]
where count($libro) > 100
return $e
```

- Restituisce gli editori con oltre 100 libri in elenco
 - ATTENZIONE:
 - la “cardinalità” del risultato, cioè il numero di editori, dipende da quante volte è eseguita la return, e questo a sua volta dipende dalle clausole for (la clausola let non influenza tale cardinalità)
 - Ogni editore “promosso” viene restituito oltre cento volte !!!

distinct-values()

- Iteriamo \$e solo sui distinti valori di Editore:

```
for $e in distinct-values( doc("libri.xml")//Editore )  
  let $libro := doc("libri.xml")//Libro[Editore = $e]  
  where count($libro) > 100  
  return $e
```
- Restituisce gli editori con oltre 100 libri in elenco
 - Ogni editore “promosso” è considerato una sola volta (si “candida” una sola volta ad essere filtrato da parte della clausola where)

Espressioni condizionali

- Estrarre, per ogni libro con almeno un autore, il titolo e i primi due autori, aggiungendo un elemento vuoto et-al se il libro ha più di due autori.

```
<risultati>
{
  for $book in doc("libri.xml")//libro
  where count($book/autore) > 0
  return <libroCompatto>
    {$book/titolo}
    {for $author in $book/autore[position()<=2]
    return $author }
    {if (count($book/autore) > 2)
    then <et-al/>
    else () }
  </libroCompatto>
}
</risultati>
```

Costruzione di strutture con attributi

- Estrarre una lista con un elemento per ogni libro, e il numero degli autori di ciascuno inserito come attributo

```
<listaLibriConNumAutori>
{
  for $libro in doc("libri.xml")//libro
  let $authors := $libro/autore
  return <libro numAutori="{ count($authors) }"/>
}
</listaLibriConNumAutori>
```

Comandi di aggiornamento

- XQuery Update permette di esprimere comandi di modifica
- Inserire come autore Italo Calvino nel libro intitolato «Il Visconte dimezzato»

insert node

```
<autore>Italo Calvino</autore>
```

```
as first into doc ("libri.xml")//libro[titolo="Il Visconte dimezzato"]
```

- Rimuovere il secondo autore dal libro intitolato «Il Visconte dimezzato»

```
delete node doc ("libri.xml")//libro[titolo="Il Visconte dimezzato"]/autore[position()=2]
```

Estensioni in XQuery 3.0

- XQuery 3.0 ha introdotto alcune novità
 - Clausola `group by`
 - Consente di esprimere raggruppamenti in modo più compatto ed efficace rispetto a XQuery 1.0
 - Opzione `allowing empty` nel `for` nidificato
 - Equivalente al join esterno (*outer join*) di SQL
 - Supporto per la gestione di flussi (*stream*) di dati

Basi di dati XML

- Due principali famiglie di sistemi
- **Basi di dati XML native**
 - Sfruttano tecnologie specifiche per XML per memorizzare e indicizzare collezioni di documenti
 - Adottano linguaggi di interrogazione specifici per XML (es: XQuery)
- **Basi di dati relazionali con supporto XML**
 - Usano il modello relazionale, esteso in modo opportuno per supportare dati XML
 - Sfruttano estensioni di SQL per l'interrogazione (es: SQL/XML)

Basi di dati XML native

- Adottano un modello **logico** dei dati non-relazionale, standard o proprietario
 - ES: DOM, XPath Data Model, XML Information Set
- Utilizzano schemi **fisici** di memorizzazione proprietari:
 - ES: metodi basati su testo (CLOB), metodi mutuati dalle basi ad oggetti
- Sono in grado di gestire tutte le caratteristiche sintattiche dei documenti (si parla di **document-centric XML data**)
 - ES: entity, ordine dei sottoelementi, commenti ecc..
- Organizzano i dati in **collezioni di documenti**, con un ruolo simile alle istanze di database dei sistemi relazionali
- Esempi di DBMS: Tamino, Xyleme, ecc..

Basi di dati relazionali con supporto XML

- Memorizzano internamente i documenti XML in tabelle
- Implicano una conversione di ingresso XML → relazionale e in uscita relazionale → XML
- Differiscono per lo schema relazionale usato per mappare i dati XML
 - Fisso, indipendente dal DTD
 - Variabile, dipendente dal DTD
- Normalmente non preservano tutte le caratteristiche sintattiche di XML

Riferimenti

- DOM: <http://www.w3.org/DOM/>
- XPath Data Model: <http://www.w3.org/TR/xpath>
- XML Infoset: <http://www.w3.org/TR/xml-infoset/>
- XQuery: <http://www.w3.org/XML/Query>
- XQuery Use Cases: <http://www.w3.org/TR/xquery-use-cases/>

- SQL/XML: J. Melton, Advanced SQL:1999, Morgan Kaufmann, 2003
- eXist, Saxon, Galax... BaseX:
 - eccellente per esercitarsi