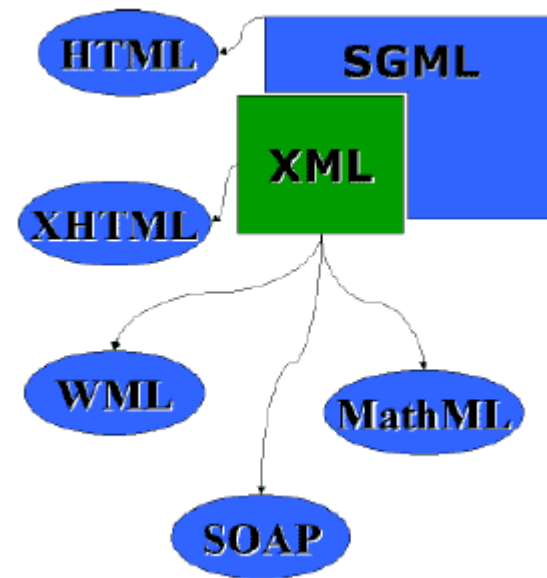


Basi di dati per XML  
Prof. Alfredo Pulvirenti  
Prof. Salvatore Alaimo

- 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).

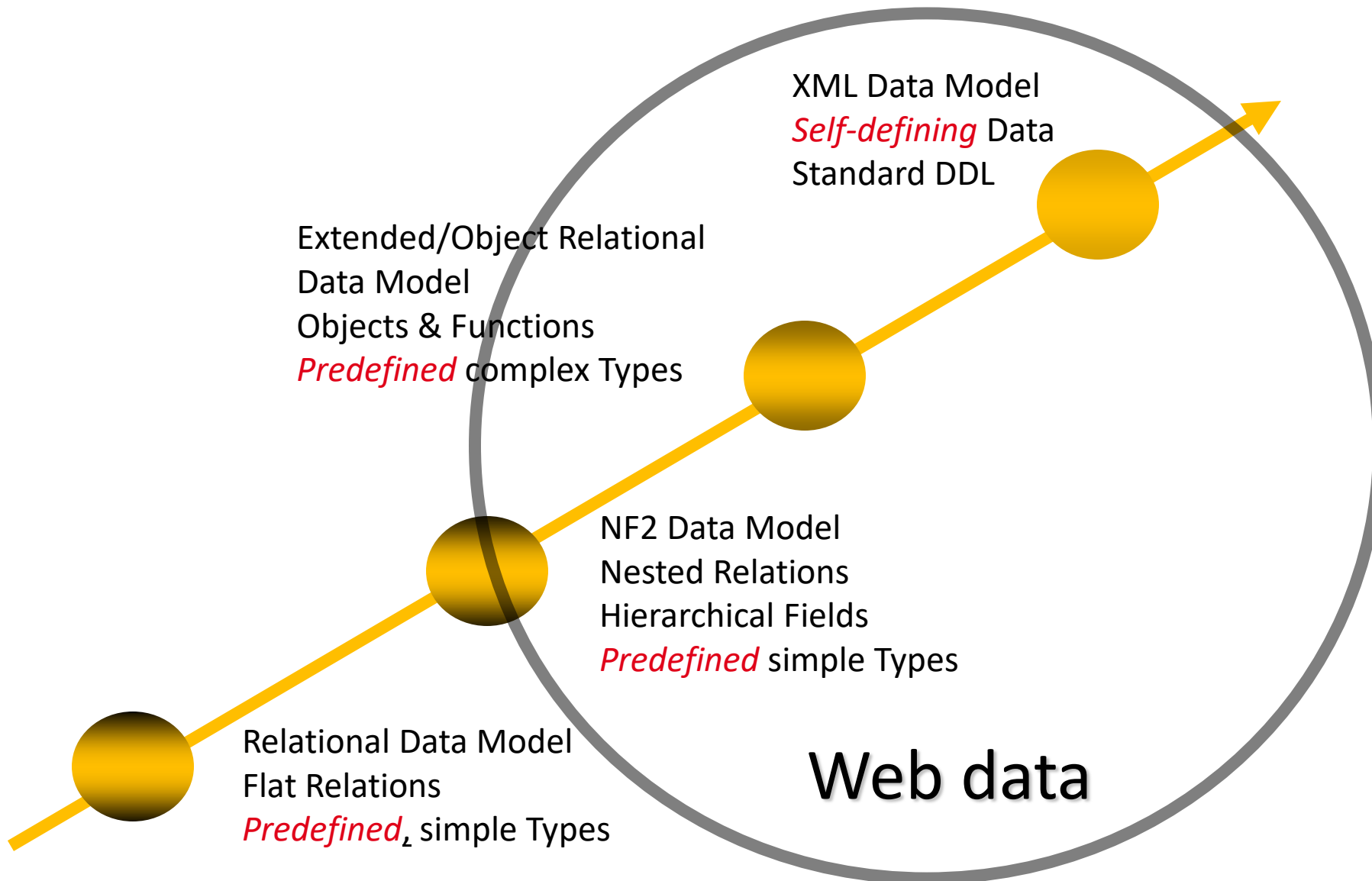


- XML è un linguaggio a marcatori estendibile.
- I linguaggi a marcatori consentono di descrivere con precisione qualsiasi tipo di informazione: gerarchica, lineare, relazionale o binaria.
- In XML l'informazione viene organizzata utilizzando una struttura gerarchica che è possibile scorrere e navigare 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 **referimenti ad entità**.
- Il valore non può contenere markup, sezioni CDATA o virgolette uguali a quelle iniziali.

# Sintassi attributi

**<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="&quot; salve &quot;">**

- 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
<code>&amp;amp;</code>	<code>&amp;</code>
<code>&amp;lt;</code>	<code>&lt;</code>
<code>&amp;gt;</code>	<code>&gt;</code>
<code>&amp;apos;</code>	<code>'</code>
<code>&amp;quot;</code>	<code>"</code>

# Grammatiche DTD

## DTD (Document Type Definition)

```
<?xml
version="1.0"
encoding="UTF-8"
standalone="yes"?>
<Corso>
  <docente>Alfredo Pulvirenti</docente>
  <titolo>TEC-23</titolo>
  <locazione>INGV Catania</locazione>
  <Descrizione>...</Descrizione>
  <Temi>
    <Tema>
      <numero>1</numero>
      <titolo>Uno sguardo al Web</titolo>
      <testo> ... </testo>
    </Tema>
    <Tema>
      <numero>2</numero>
      <titolo> Il linguaggio HTML</titolo>
      <testo> ... </testo>
    </Tema>
    <Tema>
      <numero>3</numero>
      <titolo>Linguaggi per il web server side: il PHP</titolo>
      <testo> ... </testo>
    </Tema>
    <Tema>
      <numero>4</numero>
      <titolo>XML</titolo>
      <testo> ... </testo>
    </Tema>
    <Tema>
      <numero>5</numero>
      <titolo>XHTML</titolo>
      <testo> ... </testo>
    </Tema>
    <Tema>
      <numero>6</numero>
      <titolo>Linguaggi per il web client side: Javascript</titolo>
      <testo> ... </testo>
    </Tema>
  </Temi>
</Corso>
```

```
<!DOCTYPE Corso [
  <!ELEMENT Corso (docente,titolo, locazione,Descrizione,Temi)>
  <!ELEMENT docente (#PCDATA)>
  <!ELEMENT titolo (#PCDATA)>
  <!ELEMENT Descrizione (#PCDATA)>
  <!ELEMENT Temi (Tema+)>
  <!ELEMENT Tema (numero,titolo,testo)>
  <!ELEMENT numero (#PCDATA)>
  <!ELEMENT testo (#PCDATA)>
]>
```

# Grammatiche DTD

```
<?xml version="1.0">
<!DOCTYPE Corso [
<!ELEMENT CORSO (docente,titolo, locazione,Descrizione,Temi)>
<!ELEMENT autore (#PCDATA)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT Descrizione (#PCDATA)>
<!ELEMENT Temi (Tema+)>
<!ELEMENT Tema (numero,titolo,testo)>
<!ELEMENT numero (#PCDATA)>
<!ELEMENT testo (#PCDATA)>
]>
...
```

```
<?xml version="1.0">
<!DOCTYPE Corso SYSTEM "grammatica.DTD">
...
```

- Una grammatica DTD è costituita da un insieme di regole che stabiliscono la composizione e la struttura di un documento XML.
- La grammatica può essere inserita direttamente nel documento XML, oppure in un file separato e richiamato dal documento xml

# Elementi della grammatica

- Sintassi

<!ELEMENT nome\_elemento (tipologia)>

- Informazioni testuali

```
<Indirizzo>  
Via Etnea, 10  
</Indirizzo>  
  
<!ELEMENT Indirizzo (#PCDATA) >
```

- Ulteriori elementi (elementi annidati)

<!ELEMENT elemento\_padre(f\_1,f\_2,f\_3)>

```
...  
<dati_anagrafici>  
  <nome>Mario</nome>  
  <cognome>Rossi</cognome>  
  <indirizzo>Via Etnea,10</indirizzo>  
</dati_anagrafici>
```

```
<!ELEMENT dati_anagrafici  
  (nome,cognome,indirizzo)>  
  
<!ELEMENT nome (#PCDATA)>  
<!ELEMENT cogome (#PCDATA)>  
<!ELEMENT Indirizzo (#PCDATA) >
```



# Elementi della grammatica

- Informazioni testuali ed elementi annidati
- Elementi vuoti

# Esempio completo

```
<?xml version="1.0"?>
<!DOCTYPE dati_anagrafici [
  <!ELEMENT dati_anagrafici (nome,cognome,indirizzo)>
  <!ELEMENT nome (#PCDATA)>
  <!ELEMENT cognome (#PCDATA)>
  <!ELEMENT indirizzo (#PCDATA) >
]>
<dati_anagrafici>
  <nome>Mario</nome>
  <cognome>Rossi</cognome>
  <indirizzo>Via Etnea,10</indirizzo>
</dati_anagrafici>
```

# Esempio completo

```
<?xml version="1.0"?>
<!DOCTYPE dati_anagrafici [
  <!ELEMENT dati_anagrafici (nome,cognome,indirizzo)>
  <!ELEMENT nome (#PCDATA)>
  <!ELEMENT cognome (#PCDATA)>
  <!ELEMENT indirizzo (#PCDATA) >
]>
<dati_anagrafici>
  <nome>Mario</nome>
  <cognome>Rossi</cognome>
  <indirizzo>Via Etnea,10</indirizzo>
</dati_anagrafici>
```

Oltre alla verifica della correttezza spesso è necessario riconoscere se un documento XML è conforme ad una struttura predefinita.

[http://validator.w3.org/#validate\\_by\\_input](http://validator.w3.org/#validate_by_input)

# Numero degli elementi di una grammatica

```
<!DOCTYPE dati_anagrafici [  
<!ELEMENT dati_anagrafici (nome+,cognome?,indirizzo*)>  
<!ELEMENT nome (#PCDATA)>  
<!ELEMENT cognome (#PCDATA)>  
<!ELEMENT Indirizzo (#PCDATA) >  

```

```
<!DOCTYPE dati_anagrafici [  
<!Element dati_anagrafici (nome+,cognome?,indirizzo*,  
    (codice_fiscale | partita_iva))>  
<!ELEMENT nome (#PCDATA)>  
<!ELEMENT cognome (#PCDATA)>  
<!ELEMENT Indirizzo (#PCDATA)>  
<!ELEMENT codice_fiscale (#PCDATA)>  
<!ELEMENT partita_iva (#PCDATA)>  

```

```
<!DOCTYPE dati_anagrafici [  
<!Element dati_anagrafici (nome+,cognome?,indirizzo*,  
    (codice_fiscale | partita_iva)*)>  
<!ELEMENT nome (#PCDATA)>  
<!ELEMENT cognome (#PCDATA)>  
<!ELEMENT Indirizzo (#PCDATA)>  
<!ELEMENT codice_fiscale (#PCDATA)>  
<!ELEMENT partita_iva (#PCDATA)>  

```

- Il simbolo **+** affianco all'elemento nome, indica che il campo sarà presente una o più volte (obbligatorio una volta).
- Il simbolo **\*** affianco a indirizzo indica che il campo sarà presente zero o più volte (facoltativo).
- Il simbolo **?** affianco al cognome indica che esso è facoltativo solo una volta.
- Il simbolo **|** indica che l'elemento potrà essere o codice\_fiscale o partita\_iva.
- L'ultimo esempio indica che gli elementi partita\_iva e codice\_fiscale potranno essere presenti contemporaneamente.

# Sintassi generale

<!ELEMENT name content-model>

- **name** è il nome dell' elemento da definire. Un elemento già definito non può essere ridefinito.
- **content-model** definisce gli elementi nidificabili in quello definito, e può essere:
  - EMPTY: l' elemento non può contenere **nulla**;
  - ANY: l' elemento può contenere **testo e ogni altro elemento**;
  - un **modello di contenuto**, se l' elemento può contenere solo altri elementi;
  - un **modello misto**, se l' elemento può contenere anche testo.
- **Modello di Contenuto**  
<!ELEMENT persona (titolo?, nome,cognome,(indirizzo | telefono)\*)>
- Il modello di contenuto è simile a una **espressione regolare**. **Ogni nome di elemento è anche un modello valido**. Esso indica che l' elemento definito deve contenere **esattamente un elemento del tipo dato**.
- Se  $p$  e  $q$  sono due modelli validi allora lo sono anche:
  - $(p)$       **raggruppamento** (equivale a  $p$ )
  - $p|q$       **disgiunzione** ( $p$  oppure  $q$ )
  - $p,q$       **concatenazione** ( $p$  e poi  $q$ )
  - $p^*$       **star** (zero o più volte  $p$ )
  - $p^+$       **croce** (una o più volte  $p$ )
  - $p?$       **opzione** ( $p$  oppure nulla)
- **Modelli Misti**  
<!ELEMENT testo (#PCDATA | nota)\*>
- Il **modello misto** si usa per gli elementi che **devono contenere anche testo semplice**. Il contenuto testuale si indica con **#PCDATA**.

```

<?xml version="1.0"?>
<!DOCTYPE Autore [
<!ELEMENT Autore (Nome,Cognome,pubblicazione+)>
<!ELEMENT Nome (#PCDATA)>
<!ELEMENT Cognome (#PCDATA)>
<!ELEMENT pubblicazione (titolo, abstract, tipoPub+,coautori*)>
<!ELEMENT titolo (#PCDATA)>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT tipoPub (rivista|conferenza|libro)>
<!ELEMENT coautori (#PCDATA)>
<!ELEMENT rivista (Nome,impactf,volume)>
<!ELEMENT impactf (#PCDATA)>
<!ELEMENT volume (#PCDATA)>
<!ELEMENT conferenza(titolo,luogo,data)>
<!ELEMENT luogo (#PCDATA)>
<!ELEMENT data (#PCDATA)>
<!ELEMENT libro (editore,data)>
<!ELEMENT editore (#PCDATA)>
<!ENTITY abs SYSTEM "absract.txt">
]>
<autore>
  <nome>Filippo</nome>
  <cognome>Gialli</cognome>
  <pubblicazione>
    <titolo>WEB III</titolo>
    <abstract>&abs;</abstract>
    <tipoPub>
      <libro>
        <editore>APGEO</editore>
        <data>10/10/2009</data>
      </libro>
    </tipoPub>
    <coautori>Enzo Gialli</coautori>
  </pubblicazione>

```

- Autore
  - Nome TESTO
  - Cognome TESTO
  - Pubblicazione ELEMENTO (1 o più)
    - Titolo TESTO
    - Abstract TESTO
    - TipologiaPub ELEMENT (DISGIUNTO)
      - RIVISTA ELEMENTO
      - CONFERENZA ELEMENTO
      - LIBRO ELEMENTO
      - RIVISTA
        - » NOME\_RIVISTA TESTO
        - » IMPACT FACTOR TESTO
        - » VOLUME
      - CONFERENZA
        - » TITOLO CONFERENZA TESTO
        - » LUOGO CONFERENZA TESTO
        - » DATA TESTO
      - LIBRO
        - » EDITORE TESTO
        - » DATA PUBBLICAZIONE TESTO
    - COAUTORI TESTO (0 o più)

```
<?xml version="1.0"?>
<!DOCTYPE Autore [
<!ELEMENT Autore (Nome,Cognome,pubblicazione+)>
<!ELEMENT Nome (#PCDATA)>
<!ELEMENT Cognome (#PCDATA)>
<!ELEMENT pubblicazione (abstract, tipoPub+,coautori*)>
<!ATTLIST pubblicazione titolo CDATA #REQUIRED>
<!ELEMENT abstract (#PCDATA)>
<!ELEMENT tipoPub (rivista|conferenza|libro)>
<!ELEMENT coautori (#PCDATA)>
<!ELEMENT rivista (volume)>
<!ATTLIST
  nome CDATA #REQUIRED
  impactf CDATA #IMPLIED>
<!ELEMENT volume (#PCDATA)>
<!ELEMENT conferenza(titolo,luogo,data)>
<!ELEMENT luogo (#PCDATA)>
<!ELEMENT data (#PCDATA)>
<!ELEMENT libro (editore,data)>
<!ELEMENT editore (#PCDATA)>
<!ENTITY abs SYSTEM "absract.txt">
]>
<autore>
  <nome>Filippo</nome>
  <cognome>Gialli</cognome>
  <pubblicazione
    titolo="WEB III">
    <abstract>&abs;</abstract>
    <tipoPub>
      <libro>
        <editore>APGEO</editore>
        <data>10/10/2009</data>
      </libro>
    </tipoPub>
    <coautori>Enzo Gialli</coautori>
  </pubblicazione>
</autore>
```

- Autore
  - Nome TESTO
  - Cognome TESTO
  - Pubblicazione ELEMENTO (1 o più)
    - [ATTRIBUTO DI TIPO TESTO Titolo](#)
    - Abstract TESTO
    - TipologiaPub ELEMENT (DISGIUNTO)
      - RIVISTA ELEMENTO
      - CONFERENZA ELEMENTO
      - LIBRO ELEMENTO
      - RIVISTA
        - » [ATTRIBUTO OBBLIGATORI NOME\\_RIVISTA TESTO](#)
        - » [ATTRIBUTO OPZIONALE IMPACT FACTOR TESTO](#)
        - » VOLUME
      - CONFERENZA
        - » TITOLO CONFERENZA TESTO
        - » LUOGO CONFERENZA TESTO
        - » DATA TESTO
      - LIBRO
        - » EDITORE TESTO
        - » DATA PUBBLICAZIONE TESTO
    - COAUTORI TESTO (0 o più)

- I documenti XML sono costituiti da una serie di entità. Il documento stesso è una entità.
- Tutte le entità, tranne il documento e il DTD esterno, hanno un nome.
- Le **entità parsed** sono quelle più comuni, e il parser XML le sostituisce sempre col loro testo di definizione.
- Le **entità unparsed** possono contenere **qualsiasi tipo di dato**, anche non testuale. Il parser XML non le analizza e sono accessibili solo usando le **notazioni**.



# Entità parsed

- Sono un modo pratico per inserire stringhe nel documento facendo riferimento a una definizione separata, invece di scriverle esplicitamente.
- Sono utili nel caso ci siano caratteri non digitabili direttamente, o per espandere stringhe usate di frequente, oppure per scrivere caratteri che non sono ammessi in maniera esplicita in un contesto, perché riservati (come le virgolette o i segni ‘<’ e ‘>’ ).

- Le entità vengono dichiarate all'interno della grammatica e vengono poi utilizzate nel documento XML.
- La sintassi è la seguente:  
`<!ENTITY nome definizione>`
- La definizione dell'entità è l'oggetto che viene inserito dal processore XML quando viene trovato il nome dell'entità

Esempio:

DTD

```
<!ENTITY scrittore "Joseph Conrad">  
<!ENTITY libro "La linea d' ombra">
```

XML

```
<scrittore>&scrittore;</scrittore>  
<libro>&libro;</libro>
```

# Entità esterne

## Sintassi

- Oltre alle entità interne ci sono le entità esterne. Queste non sono contenute direttamente nel documento XML ma si trovano all'interno di documenti

<!ENTITY nome SYSTEM uri>

Es.

DTD

...

<!ELEMENT titolo (#PCDATA)>

<!ELEMENT autore (#PCDATA)>

<!ELEMENT abstract (#PCDATA)>

...

<!ENTITY abstract SYSTEM "abstract.txt">

XML

<abstract>&abstract;</abstract>

# DTD: Attributi

- Gli attributi permettono di completare gli elementi
- associando ad essi alcuni semplici dati aggiuntivi.
- Con DTD possono essere definiti usando una dichiarazione del tipo `<!ATTLIST ...>`, la struttura è la seguente:

```
<!ATTLIST nome_elemento  
    nome_attributo1 tipo_attributo1 default1  
    nome_attributo2 tipo_attributo2 default2  
    .  
    .  
    .  
>
```

- I possibili valori per i default sono:

**#REQUIRED**

**#IMPLIED**

**#FIXED** valore fisso

**Default**

- Quando un attributo è dichiarato come **#REQUIRED** allora deve apparire obbligatoriamente nell'elemento in cui è specificato.
- Se invece è dichiarato **#IMPLIED** invece è opzionale.
- Attraverso la parola chiave **#FIXED** può essere specificato un valore fisso che deve assumere l'attributo, che, nel caso di omissione nel documento XML, viene assunto automaticamente dal parser. Nel caso in cui venga omessa la parola chiave **#FIXED**, ma venga indicato solo un valore di Default (ultimo caso in tabella), allora l'attributo potrà assumere anche valori diversi da quello specificato, se invece non compare nel documento il parser utilizzerà quello indicato.

```
<!DOCTYPE Corso [  
  <!ELEMENT Corso (Descrizione,Temi)>  
  <!ATTLIST Corso  
    docente CDATA #REQUIRED  
    titolo CDATA #REQUIRED  
    locazione CDATA "Laboratorio PC Wyeth" #FIXED  
  >  
  ...
```

# DTD: tipi di dati per gli attributi

- Il tipo di dato può assumere i seguenti valori:

CDATA
ENTITY
ENTITIES
ID
IDREF
IDREFS
NMTOKENS
NOTATION
Enumerated

## CDATA

valore dell' attributo di tipo testo.

## ID IDREF IDREFS

sono utili per specificare identificatori.

- ID

il valore da esso assunto deve essere unico all'interno dello stesso documento XML. Questo è utile quando si vuole essere sicuri dell'unicità del dato inserito nell'attributo (es. Codice fiscale o matricola di un dipendente).

- IDREF

utili per indirizzare attributi di tipo ID definiti da qualche altra parte nel documento.

- Es.

<!ELEMENT Prestito (nome, cognome)>

<!ATTLIST Prestito collocazione IDREF #REQUIRED >

Prestito può fare riferimento alla collocazione del libro senza dover ripetere tutto l'elemento libro.



# Esempio

```
<!DOCTYPE TVSCHEDULE [  
  
  <!ELEMENT TVSCHEDULE (CHANNEL+)>  
  <!ELEMENT CHANNEL (BANNER, DAY+)>  
  <!ELEMENT BANNER (#PCDATA)>  
  <!ELEMENT DAY (DATE, (HOLIDAY| PROGRAMSLOT+)+)>  
  <!ELEMENT HOLIDAY (#PCDATA)>  
  <!ELEMENT DATE (#PCDATA)>  
  <!ELEMENT PROGRAMSLOT (TIME, TITLE, DESCRIPTION?)>  
  <!ELEMENT TIME (#PCDATA)>  
  <!ELEMENT TITLE (#PCDATA)>  
  <!ELEMENT DESCRIPTION (#PCDATA)>  
  
  <!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>  
  <!ATTLIST CHANNEL CHAN CDATA #REQUIRED>  
  <!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>  
  <!ATTLIST TITLE RATING CDATA #IMPLIED>  
  <!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>  

```

# Esempio

```
<!DOCTYPE NEWSPAPER [  
  
  <!ELEMENT NEWSPAPER (ARTICLE+)>  
  <!ELEMENT ARTICLE (HEADLINE,BYLINE,LEAD,BODY,NOTES)>  
  <!ELEMENT HEADLINE (#PCDATA)>  
  <!ELEMENT BYLINE (#PCDATA)>  
  <!ELEMENT LEAD (#PCDATA)>  
  <!ELEMENT BODY (#PCDATA)>  
  <!ELEMENT NOTES (#PCDATA)>  
  
  <!ATTLIST ARTICLE AUTHOR CDATA #REQUIRED>  
  <!ATTLIST ARTICLE EDITOR CDATA #IMPLIED>  
  <!ATTLIST ARTICLE DATE CDATA #IMPLIED>  
  <!ATTLIST ARTICLE EDITION CDATA #IMPLIED>  
  
  <!ENTITY NEWSPAPER "Vervet Logic Times">  
  <!ENTITY PUBLISHER "Vervet Logic Press">  
  <!ENTITY COPYRIGHT "Copyright 1998 Vervet Logic Press">  
  

```

# Esempio

```
<!DOCTYPE CATALOG [  
  <!ENTITY AUTHOR "John Doe">  
  <!ENTITY COMPANY "JD Power Tools, Inc.">  
  <!ENTITY EMAIL "jd@jd-tools.com">  
  
  <!ELEMENT CATALOG (PRODUCT+)>  
  
  <!ELEMENT PRODUCT  
    (SPECIFICATIONS+,OPTIONS?,PRICE+,NOTES?)>  
  <!ATTLIST PRODUCT  
    NAME CDATA #IMPLIED  
    CATEGORY (HandTool|Table|Shop-Professional) "HandTool"  
    PARTNUM CDATA #IMPLIED  
    PLANT (Pittsburgh|Milwaukee|Chicago) "Chicago"  
    INVENTORY (InStock|Backordered|Discontinued) "InStock">  
  
  <!ELEMENT SPECIFICATIONS (#PCDATA)>  
  <!ATTLIST SPECIFICATIONS  
    WEIGHT CDATA #IMPLIED  
    POWER CDATA #IMPLIED>  
  
  <!ELEMENT OPTIONS (#PCDATA)>  
  <!ATTLIST OPTIONS  
    FINISH (Metal|Polished|Matte) "Matte"  
    ADAPTER (Included|Optional|NotApplicable) "Included"  
    CASE (HardShell|Soft|NotApplicable) "HardShell">  
  
  <!ELEMENT PRICE (#PCDATA)>  
  <!ATTLIST PRICE  
    MSRP CDATA #IMPLIED  
    WHOLESALE CDATA #IMPLIED  
    STREET CDATA #IMPLIED  
    SHIPPING CDATA #IMPLIED>  
  
  <!ELEMENT NOTES (#PCDATA)>  
  
>
```

# 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.

```
<prefisso:elemento
  prefisso:attributo1="valore"
  prefisso:attributo2="valore"
  ...
Contenuto elemento
</prefisso:elemento>
```

Anche nella grammatica DTD possiamo fare riferimento ai namespace:

```
<!ELEMENT clienti(cliente+)>
<!ATTLIST clienti
  xmlns:ct CDATA #REQUIRED
  xmlns:rg CDATA #REQUIRED
  xmlns   CDATA #REQUIRED
>
<!ELEMENT cliente (ragione_sociale,partita_iva,ct:indirizzo,email, citta,rg:indirizzo,telefono)
>
<!ELEMENT ct:indirizzo (#PCDATA)>
<!ATTLIST ct:indirizzo
  ct:sedi (corso|via|piazza) #REQUIRED
>
<!ELEMENT ..
>
```

# XML Schema

Storia: - inizialmente proposto da Microsoft

- divenuto W3C recommendation (maggio '01)

Scopo: definire gli elementi e la composizione di un documento XML in modo più preciso del DTD

Un XML Schema definisce regole riguardanti:

- Elementi
- Attributi
- Gerarchia degli elementi
- Sequenza di elementi figli
- Cardinalità di elementi figli
- Tipi di dati per elementi e attributi
- Valori di default per elementi e attributi

- **XML Schema** = insieme di elementi XML standard per definire schemi di documenti detti **XSD = XML Schema Definition**: schema di un tipo di documenti
- Gli XSD sono:
  - Estendibili (ammettono tipi riusabili definiti dall'utente)
  - In formato XML
  - Più ricchi e completi dei DTD
  - Capaci di supportare tipi di dati diversi da PCDATA
  - Capaci di gestire namespace multipli

# Documento XML con riferimento a XSD

```
<?xml version="1.0"?>
<note
xmlns="http://www.w3schools.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation=http://www.w3schools.com/note.xsd" >
    <to> Tove </to>
    <from> Jani </from>
    <head> Reminder </head>
    <body> Don't forget me this weekend! </body>
</note>
```

Note:

**xmlns:** namespace di default

**xmlns:xsi:** URI (Universal Resource Identifier) che introduce l'uso dei tag di XML Schema

**xsi:schemaLocation:** dichiara dove reperire il file XSD (sempre attraverso URI)

# Esempio di XML Schema

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="note">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="to" type="xs:string"/>
        <xs:element name="from" type="xs:string"/>
        <xs:element name="head" type="xs:string"/>
        <xs:element name="body" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

**xs:** namespace per XSchema  
(contiene tutti i tag XSD)

# Elementi semplici (Simple elements)

- Possono contenere solo testo (no elem. o attributi)

- Forme di dichiarazione:

```
<xs:element name="nome" type="tipo"/>
```

```
<xs:element name="nome" type="tipo" default="xyz" />
```

```
<xs:element name="nome" type="tipo" fixed="xyz" />
```

- Tipi: stringhe, numerici, date, time, Boolean, ecc..
- Esempi di definizione di elementi semplici in XSD

```
<xs:element name="età" type="xs:integer"/>
```

```
<xs:element name="cognome" type="xs:string"/>
```

- Esempi di elementi semplici XML

```
<età> 65 </età>
```

```
<cognome> Rossi </cognome>
```

- Definizione di attributi

```
<xs:attribute name="name" type="type"/>
```

```
<xs:attribute ... default|fixed="xyz"  
    use="required|optional"/>
```

- Esempio di definizione di attribute

```
<xs:attribute name="lang" type="xs:string"/>
```

- Esempio di uso di un attributo

```
<lastname lang="it"> qwerty </lastname>
```



- Consentono di dichiarare vincoli sui valori di un tipo elementare (detti tipi base)

```
<xs:element name="age">  
  <xs:simpleType>  
    <xs:restriction base="xs:integer">  
      <xs:minInclusive value="0"/>  
      <xs:maxInclusive value="100"/>  
    </xs:restriction>  
  </xs:simpleType>  
</xs:element>
```

# Enumerazione

- Particolare tipo di restrizione che consente di enumerare i valori di un elemento o attributo

```
<xs:element name="car">  
  <xs:simpleType>  
    <xs:restriction base="xs:string">  
      <xs:enumeration value="Audi"/>  
      <xs:enumeration value="FCA"/>  
      <xs:enumeration value="BMW"/>  
    </xs:restriction>  
  </xs:simpleType></xs:element>
```

# Elementi complessi (Complex elements)

- Esistono quattro tipi di elementi complessi:
  - Vuoti (empty)
  - Contenenti solo altri elementi
  - Contenenti solo testo
  - Contenenti testo e/o altri elementi
- Sintassi per definire elementi complessi:

```
<xs:element name="name">  
  <xs:complexType>  
    .. element content ..  
  </xs:complexType>  
</xs:element>
```

# Costrutto sequence

- Sequenza (= record): gli elementi devono apparire nell'ordine indicato

```
<xs:element name="libro">
```

```
<xs:complexType>
```

```
<xs:sequence>
```

```
<xs:element name="titolo" type="xs:string"/>
```

```
<xs:element name="autore" type="xs:string"  
    maxOccurs="unbounded"/>
```

```
</xs:sequence>
```

```
</xs:complexType>
```

```
</xs:element>
```

- La sequenza può contenere sia sottoelementi che attributi

# Costrutto sequence con attributi

```
<xs:element name="libro">  
  <xs:complexType>  
    <xs:sequence>  
      <xs:element name="titolo" type="xs:string"/>  
      <xs:element name="autore" type="xs:string"  
        maxOccurs="unbounded"/>  
    </xs:sequence>  
    <xs:attribute name="editore" type="xs:string"  
      use="required"/>  
    <xs:attribute name="pubblicazione" type="xs:date"/>  
  </xs:complexType>  
</xs:element>
```

# Specifica del contenuto: costruito all

- Come la sequenza ma gli elementi possono apparire **nel documento in qualsiasi ordine**

```
<xs:element name="libro">
```

```
<xs:complexType>
```

```
<xs:all>
```

```
<xs:element name="titolo" type="xs:string"/>
```

```
<xs:element name="autore" type="xs:string"  
    maxOccurs="unbounded"/>
```

```
</xs:all>
```

```
</xs:complexType>
```

```
</xs:element>
```

# Costrutto choice

- Sequenza (= OR)

```
<xs:element name="parte">
```

```
<xs:complexType>
```

```
<xs:choice>
```

```
<xs:element name="capitolo" type="xs:string"  
    maxOccurs="unbounded"/>
```

```
<xs:element name="appendice" type="xs:string"  
    maxOccurs="unbounded"/>
```

```
</xs:choice>
```

```
</xs:complexType>
```

```
</xs:element>
```

- Gli elementi interni appaiono in alternativa nel documento

- Gli elementi senza sottoelementi interni si dichiarano come complex type privi di sottoelementi

```
<xs:element name="product">  
  <xs:complexType>  
    <xs:attribute name="prodid" type="xs:integer"/>  
  </xs:complexType>  
</xs:element>
```



# Elementi MIXED

```
<xs:element name="letter">
  <xs:complexType mixed="true">
    <xs:sequence>
      <xs:element name="name" type="xs:string"/>
      <xs:element name="orderid" type="xs:integer"/>
      <xs:element name="shipdate" type="xs:date"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

- **Esempio**

```
<letter> Dear Mr.<name>John Smith</name>. Your order
  <orderid>1032</orderid> will be shipped on <shipdate>2001-07-
  13</shipdate>.
</letter>
```

# Specifica della cardinalità

- Indica la cardinalità dei sotto-elementi
- La sintassi prevede l'uso di due attributi:
  - **maxOccurs**: max numero di occorrenze
  - **minOccurs**: min numero di occorrenze
- Se non specificati: default = 1

```
<xs:element name="person">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="full_name" type="xs:string"/>
      <xs:element name="child_name" type="xs:string"
        minOccurs="0" maxOccurs="10"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

# Definizioni di tipi riusabili

- Consente di definire tipi e riusarli per:
  - Tipare attributi
  - Definire altri tipi
- **ATTENZIONE:** Definizione di tipo, non di elemento

```
<xs:complexType name="personinfo">
```

```
<xs:sequence>
```

```
<xs:element name="firstname" type="xs:string"/> <xs:element name="lastname" type="xs:string"/>
```

```
</xs:sequence>
```

```
</xs:complexType>
```

- Riutilizzo del tipo, per definire elementi/attributi

```
<xs:element name="employee" type="personinfo"/>
```

```
<xs:element name="student" type="personinfo"/>
```

```
<xs:attribute name="member" type="personinfo"/>
```

# Definizione di gruppi di elementi

- Consente di definire blocchi di elementi e riusarli per definire altri tipi o elementi (si può fare anche con attributi, usando **attributeGroup**)

**<xs:group name="custGroup">** Definizione di gruppo riusabile

**<xs:sequence>**

**<xs:element name="customer" type="xs:string"/>**

**<xs:element name="orderdetails" type="xs:string"/>**

**<xs:element name="billto" type="xs:string"/>**

**</xs:sequence>**

**</xs:group>**

**<xs:element name="order" type="ordertype"/>**

**<xs:complexType name="ordertype">** Riuso di “custGroup”

**<xs:group ref="custGroup"/>**

**<xs:attribute name="status" type="xs:string"/>**

**</xs:complexType>**

# Riferimenti

- [www.w3schools.com](http://www.w3schools.com)
- [www.w3schools.com/xml](http://www.w3schools.com/xml)
- [www.w3schools.com/dtd](http://www.w3schools.com/dtd)
- <http://www.w3schools.com/schema/>

- XSL: eXtensible Style Language
- E' una componente di un linguaggio per la definizione di fogli di stile che consentono di gestire la presentazione di dati contenuti in documenti XML in modo da poterli visualizzare su browser.
- XSLT è un sw che prende un documento XSLT e lo applica ad un documento XML producendo un altro documento come risultato.

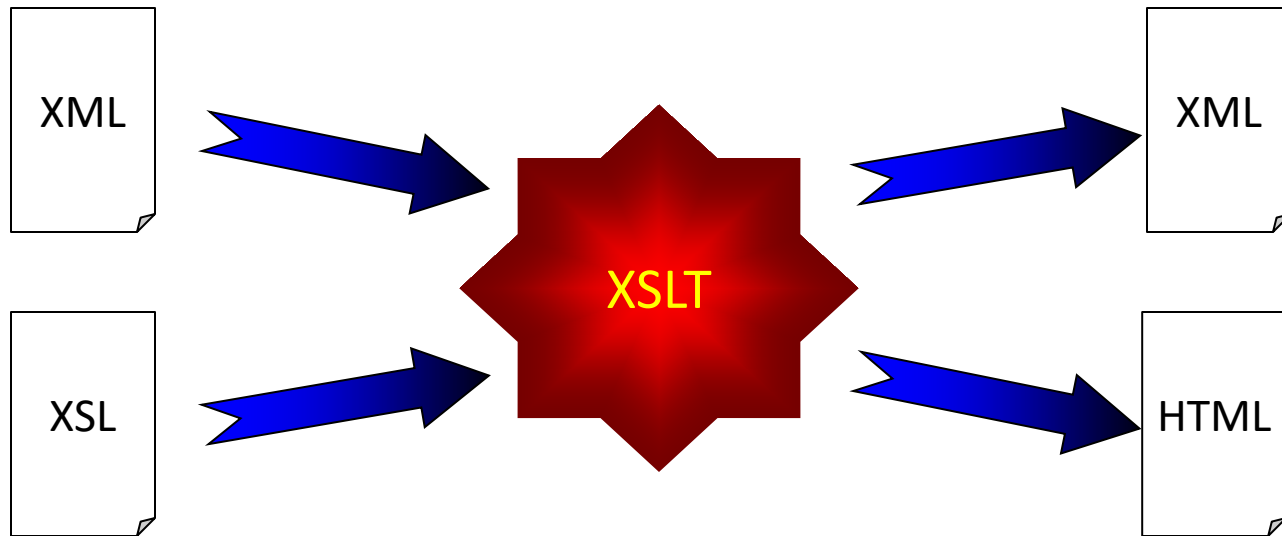
# Fogli di stile

- **1996: DSSSL** - standard che definisce fogli di stile per SGML (ISO/IEC 10179:1996)
- **1996-1998: CSS** - fogli di stile per HTML - sono attualmente disponibili due specifiche: CSS1 e CSS2 (approvati dal W3C rispettivamente nel Dicembre 1996 e nel Maggio 1998)
- **Agosto 1997:** Microsoft, Arbortext e Inso Corp. sottopongono la specifica **XSL** al W3C (sottoinsieme di DSSSL) per dati XML altamente strutturati - è diventato standard nel 1999

- Linguaggio per rappresentare l'output di **XML formattato** e per **convertire** un documento XML in un altro documento (XML, HTML, o qualunque formato)
- Usa la **notazione XML**
- **XSL = XSLT + XSL FO**
- XSLT (XSL Transformation) **Trasforma** un documento XML in un altro documento XML o altro tipo di documento (HTML, ecc.)
- Può:
  - aggiungere nuovi elementi;
  - rimuovere elementi presenti; riorganizzare gli elementi;
  - decidere quali visualizzare, ecc.
- XSL FO (Formatting Object) contiene le istruzioni per **formattare** l'output di un documento XML



# XSLT



- Utilizza **XPath** per definire le parti del documento sul quale effettuare le trasformazioni
- Per gli elementi sui quali devono essere applicate le trasformazioni vengono definiti dei **template**

# Template

- Per assegnare uno stile ad un particolare elemento XML oppure per applicare delle trasformazioni si usa un **template** nel foglio di stile
- Il foglio di stile può essere eseguito da un **processore XSLT** che scandisce il documento XML sorgente, identifica gli elementi per i quali è stato definito un template nel foglio di stile, ed effettua le azioni specificate nel template.

# Esempio di template (1)

```
<xsl:template match=paragrafo>
```

```
...
```

```
<xsl:template>
```

- La clausola **match** definisce su quali elementi si applica il template
- Per ogni elemento si possono specificare più template (si applica il più specifico oppure si assegna una priorità per l'applicazione)
- Un template può specificare lo stile per più elementi

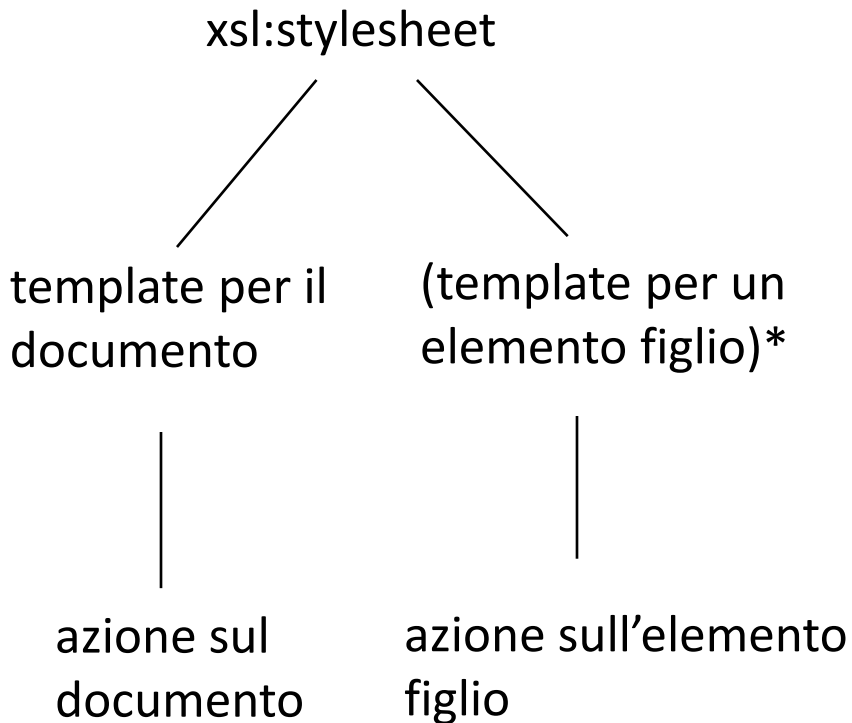
# Esempio di template (2)

```
<xsl:template match=paragrafo>  
  <fo:block font-size="10pt" space-before="12pt">  
    <xsl:apply-templates/>  
  </fo:block>  
</xsl:template>
```

- All'interno del template si specifica come si devono **processare** gli elementi figli (**xsl:apply-templates**)
- Nell'esempio: gli elementi "paragrafo" diventano oggetti di formattazione "blocco" che vengono visualizzati con 10 punti e 12 punti di spazio dopo ogni blocco

- Si possono specificare gli elementi figli da processare utilizzando i seguenti simboli:
- `|` operatore or
- `.` Elemento corrente
- `//` discendenti
- `/` figli
- `..` Padre
- `@` identifica un attributo
- `first-of-any()`, `first-of-type()`
- `last-of-any()`, `last-of-type()`

# Struttura di un documento XSL



```
<?xml version="1.0"?>
<xsl:stylesheet>

  <xsl:template match="/">
    [azione]
  </xsl:template>

  <xsl:template match="Elenco">
    [azione]
  </xsl:template>

  <xsl:template match="Libro">
    [azione]
  </xsl:template>
  ...

</xsl:stylesheet>
```

# Intestazione

- Il foglio di stile è interamente contenuto in un tag di tipo `xsl:stylesheet`, e inizia con una intestazione come segue:

```
<xsl:stylesheet  
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">  
</xsl:stylesheet>
```

- Possiamo riconoscere due dichiarazioni: spazio dei nomi e numero di versione.
- Lo spazio dei nomi è obbligatorio, e deve utilizzare l'URL mostrato. Serve a distinguere il particolare formato XSLT utilizzato.
- Il numero di versione serve a distinguere un foglio di stile che usa un certo standard da eventuali future revisioni. Tutte le regole di trasformazione sono contenute nel corpo di questo tag.



# match specifica il nodo da sostituire

- L'attributo match usa una espressione XPath per specificare a quali nodi si deve applicare. Nella maggior parte dei casi seleziona solo uno specifico tipo di nodo.
- Le espressioni XPath sono relative al nodo corrente.
- Quindi indicare solamente un tipo di nodo permette di selezionare uno dei figli del nodo corrente.
- In ogni caso è abbastanza frequente specificare nodi che non sono figli del nodo corrente, come pure è possibile selezionare attributi o altri elementi del documento XML.

# Esempio

- ```
<xsl:template match="/">  
  <html>  
    <body bgcolor="#ffffff">  
      <table border="0" cellpadding="0">  
        <xsl:apply-templates select="*" />  
      </table>  
    </body>  
  </html>  
</xsl:template>
```
- Questa regola serve a iniziare la trasformazione di documento.
- Il nodo radice viene trasformato in un documento HTML con lo sfondo bianco che contiene una tabella.

# Mode, modalità di processing

- L'attributo mode permette di applicare un template in casi diversi.
- La modalità viene specificata da apply-templates. Così si possono avere due template con la stessa espressione di match (si riferiscono agli stessi nodi) ma usati in casi distinti.
- Per esempio, un template per costruire l'indice e un altro per il corpo del documento, ma che si applicano agli stessi in momenti diversi.

# Name, nome di un template

- Un template può avere un nome e può venire chiamato esplicitamente utilizzando `xsl:call-template`, per di più specificando dei parametri.

- Definizione

```
<xsl:template name="row">  
    <tr><td> <xsl:value-of select="."/> </td></tr>  
</xsl:template>
```

- Uso

```
<xsl:call-template name="row"/>
```

## xsl:apply-templates

- A xsl:apply-templates si ricorre all'interno di un template. Serve a selezionare i nuovi nodi che devono essere aggiunti alla lista corrente. Se non viene specificato alcun attributo vengono selezionati tutti i nodi figli del nodo corrente.
- Si specifica con l'attributo select, usando XPath, quali nodi devono essere ulteriormente elaborati.
- Non necessariamente sono figli ma posso essere i nipoti oppure degli antenati, o ancora possono essere trattati in maniera particolare gli attributi.

# xsl:for-each

- ```
<xsl:template match="chap">  
  <xsl:for-each select="p">  
    <xsl:value-of select=".">  
  </xsl:for-each>  
</xsl:template>
```

equivale a (senza una regola)

```
<xsl:template match="chap">  
  <xsl:apply-templates select="p"/> </xsl:templates>  
<xsl:template match="p">  
  <xsl:value-of select="."/> </xsl:templates>
```

## xsl:value-of

- È infatti possibile utilizzare delle espressioni che calcolano l'output.

```
<xsl:template match="ragione_sociale">  
<td><xsl:value-of select="."/></td> </xsl:template>
```

- Il corpo della regola non contiene altri tag ma semplicemente xsl:value-of. Questo comando serve a produrre un output calcolato.

- Nell'esempio si seleziona il nodo corrente (.).
- Le regole per il calcolo del valore di un nodo (valore che viene inviato nell'output) sono abbastanza intuitive:
  - Il valore di un nodo testo è uguale al testo stesso.
  - Il valore di un nodo tag è uguale alla concatenazione dei valori dei nodi contenuti.



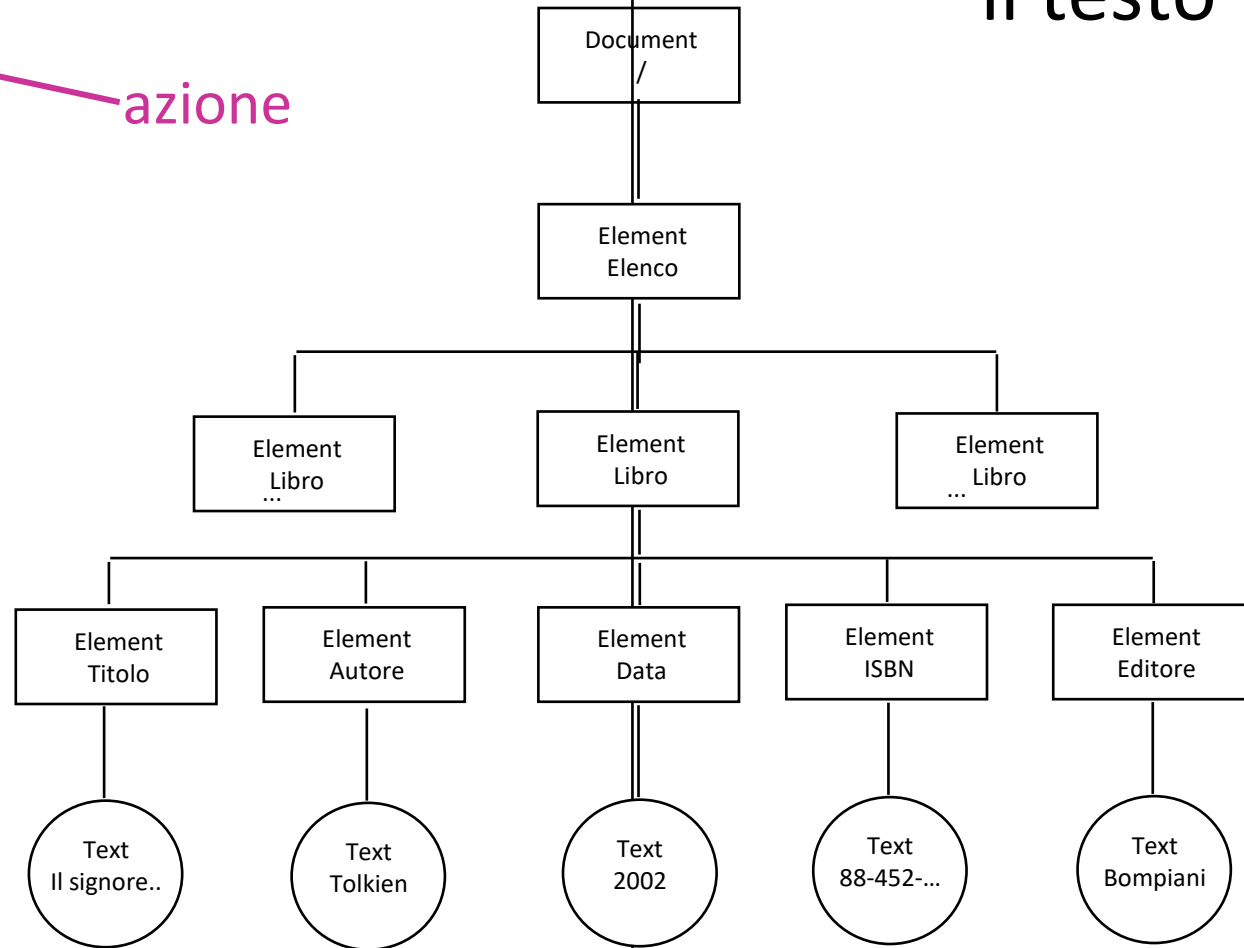
# value template

- Gli attributi in output hanno una prerogativa: possono essere calcolati senza utilizzare `xsl:value-of` (*attribute value template*)
- Negli attributi si possono inserire espressioni tra graffe.  
`value="{/document/@title}"`
- Il valore dell'attributo non è scritto esplicitamente ma calcolato in base all'espressione tra graffe. Ciò che è fuori dalle graffe è considerato testo normale (copiato così com'è) mentre l'espressione tra graffe viene calcolata, come la select di `xsl:value-of`.
- L'espressione dell'esempio imposta l'attributo value uguale all'attributo title del document.

- Fin qui abbiamo costruito l'output semplicemente copiando i tag che non fanno parte dell'XSLT.
- Ma in generale possiamo creare tag arbitrari, con nomi derivati dall'input. Per fare questo usiamo una serie di comandi che permettono di comporre a volontà l'output.

# Estrarre il testo

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">
  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="Elenco">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="Libro">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="Titolo">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="Autore">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="Data">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="ISBN">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="Editore">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="text()">
    <xsl:value-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```



# Trasformazione

```
<?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>
```



Il Signore degli Anelli J.R.R. Tolkien  
2002 88-452-9005-0 Bompiani  
Il nome della rosa Umberto Eco  
1987 55-344-2345-1 Bompiani  
Il sospetto F. Dürrenmatt 1990  
88-07-81133-2 Feltrinelli

# Creare un documento HTML

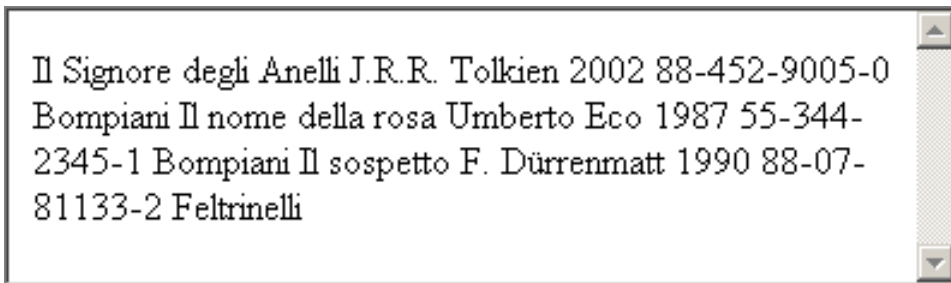
```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  <xsl:template match="/">
    <HTML>
    <HEAD>
    <TITLE>Elenco libri</TITLE>
    </HEAD>
    <BODY>
      <xsl:apply-templates/>
    </BODY>
  </HTML>
</xsl:template>
  <xsl:template match="Elenco">
    <xsl:apply-templates/>
  </xsl:template>
  <xsl:template match="Libro">
    <xsl:apply-templates/>
  </xsl:template>
  ...
</xsl:stylesheet>
```

# Trasformazione

```
<?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>
```



```
<HTML>
<HEAD>
<TITLE>Book Catalogue</TITLE>
</HEAD>
<BODY>
Il Signore degli Anelli J.R.R. Tolkien
2002 88-452-9005-0 Bompiani
Il nome della rosa Umberto Eco
1987 55-344-2345-1 Bompiani
Il sospetto F. Dürrenmatt 1990
88-07-81133-2 Feltrinelli
</BODY>
</HTML>
```



```
Il Signore degli Anelli J.R.R. Tolkien 2002 88-452-9005-0
Bompiani Il nome della rosa Umberto Eco 1987 55-344-
2345-1 Bompiani Il sospetto F. Dürrenmatt 1990 88-07-
81133-2 Feltrinelli
```

# Esempio: creare una tabella

```
<HTML>
<HEAD><TITLE>Elenco libri</TITLE></HEAD>
<BODY>
<TABLE BORDER="1" WIDTH="100%">
  <TR> <TD> Il Signore degli Anelli </TD>
    <TD> J.R.R. Tolkien </TD>
    <TD> 2002 </TD>
    <TD> 88-452-9005-0 </TD>
    <TD> Bompiani </TD>
  </TR>
  <TR> <TD> Il nome della rosa </TD>
    <TD> Umberto Eco </TD>
    <TD> 1987 </TD>
    <TD> 55-344-2345-1 </TD>
    <TD> Bompiani </TD>
  </TR>
  <TR> <TD> Il sospetto </TD>
    <TD> F. Dürrenmatt </TD>
    <TD> 1990 </TD>
    <TD> 88-07-81133-2 </TD>
    <TD> Feltrinelli </TD>
  </TR>
</TABLE>
</BODY>
</HTML>
```

Il Signore degli Anelli	J.R.R. Tolkien	2002	88-452-9005-0	Bompiani
Il nome della rosa	Umberto Eco	1987	55-344-2345-1	Bompiani
Il sospetto	F. Dürrenmatt	1990	88-07-81133-2	Feltrinelli

# Esempio: creare una tabella

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  <xsl:template match="Elenco">
    <HTML>
    <HEAD>
    <TITLE>Elenco libri</TITLE>
    </HEAD>
    <BODY>
      <TABLE BORDER="1" WIDTH="100%">
        <xsl:apply-templates/>
      </TABLE>
    </BODY>
    </HTML>
  </xsl:template>
  <xsl:template match="Libro">
    <TR>
      <xsl:apply-templates/>
    </TR>
  </xsl:template>
  <xsl:template match="Titolo | Autore | Data | ISBN | Editore">
    <TD>
      <xsl:apply-templates/>
    </TD>
  </xsl:template>
</xsl:stylesheet>
```



# Trasformazione XML-XML

Il nuovo elenco per ogni libro contiene solo il loro titolo e, se questo è disponibile, anche la sua casa editrice.

```
<?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>
```



```
<?xml version="1.0"?>
<NuovoElenco>
  <Libro>
    <Titolo>Il Signore degli Anelli</Titolo>
    <Editore>Bompiani</Editore>
  </Libro>
  <Libro>
    <Titolo>Il nome della rosa</Titolo>
  </Libro>
  <Libro>
    <Titolo>Il sospetto</Titolo>
    <Editore>Feltrinelli</Editore>
  </Libro>
</NuovoElenco>
```

# Trasformazione XML-XML

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
<xsl:template match="/">
  <NuovoElenco>
    <xsl:for-each select="Elenco/Libri">
      <Libro>
        <Titolo><xsl:value-of select="Titolo"/> </Titolo>
        <xsl:if test="@disponibilità = 'S'">
          <Editore><xsl:value-of select="Editore"/></Editore>
        </xsl:if>
      </Libro>
    </xsl:for-each>
  </NuovoElenco>
</xsl:template>
```

# Associare un foglio di stile ad un documento XML

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
  href="file:///localhost/EsempiXML-XSL/libri.xsl"?>
<!DOCTYPE Elenco SYSTEM
  "file:///localhost/EsempiXML-XSL/libri.dtd">
<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>
  ...
</Elenco>
```

- 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
```

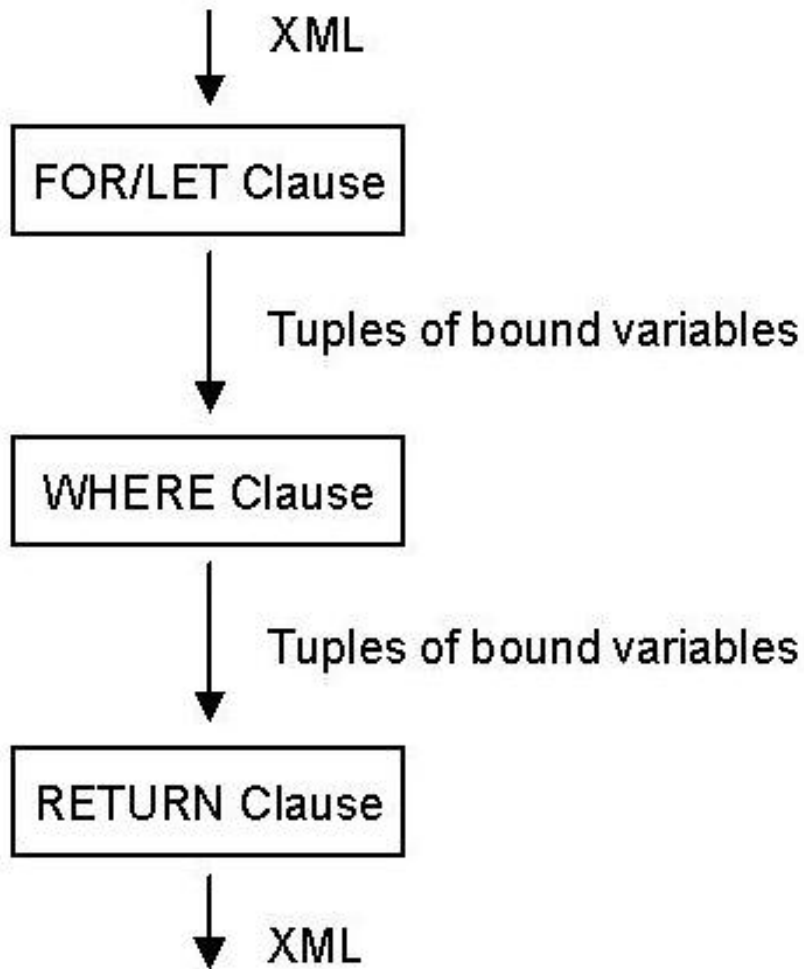
# Espressioni LET

```
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