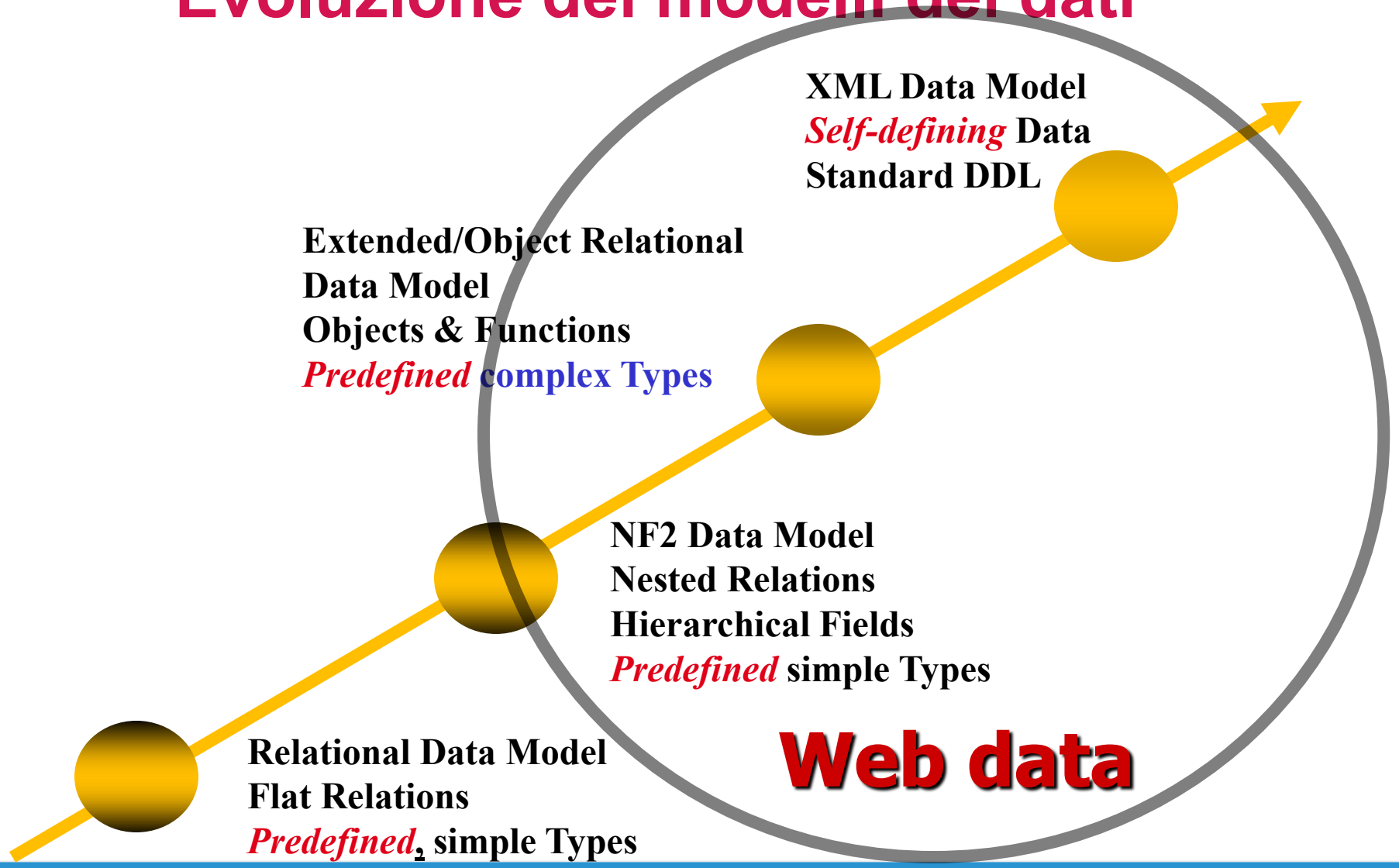
Basi Di Dati e di conoscenza

Evoluzione dei linguaggi, dei modelli e dei sistemi per basi di dati

XML-Xpath-XQuery

Evoluzione dei modelli dei dati



XML

- **Concezione originale**: un meta-linguaggio per la specifica di linguaggi di markup
- **Come in HTML**
  - I dati XML sono documenti
  - Le proprietà dei dati sono espresse mediante marcatura del contenuto
- **Come nelle basi di dati**
  - Esiste un modello dei dati (DTD, XSD)
  - Esistono linguaggi di query e trasformazione (XQuery, XSL)
  - I dati sono neutri rispetto alle modalità di rappresentazione

Evoluzione

- 1986: Standard Generalized Markup Language (SGML) ISO 8879-1986
- Novembre 1995: HTML 2.0
- Gennaio 1997: HTML 3.2
- Agosto 1997: XML W3C Working Draft
- Feb 10, 1998: XML 1.0 Recommendation
- Dec. 13, 2001: XML 1.1 W3C Working Draft
- Oct. 15, 2002 : XML 1.1 W3C Candidate Recommendation
- 2003 : XML 1.1 Standard

XML

- HTML: insieme fisso di tag
- XML: standard per creare linguaggi di markup con tag personalizzati (erede di SGML); possono essere usati in qualunque dominio
- HTML vs XML

```
<h1>The Idea
    Methodology</h1><br>
<ul>
 <li>di S. Ceri,
     P. Fraternali  </li>
 <li> Addison-Wesley</li>
 <li> US$ 49 </li>
</ul>
```

```
<book>
<title>The Idea
       Methodology </title>
 <author> S. Ceri </author>
 <author> P. Fraternali
                  </author>
 <ed> Addison-Wesley </ed>
 <price> US$ 49 </price>
</book>
```

Esempio di documento XML

```
<?xml version="1.0"?>
<elenco>
 <prodotto codice="123kl14">
    <descrizione> Forno </descrizione>
    <prezzo> 1040000 </prezzo>
 </prodotto>
 <prodotto codice="432sd35">
    <descrizione> Frigo </descrizione>
</prodotto>
</elenco>
```

XML vs HTML

- XML non rimpiazza HTML!

- XML e HTML sono nati con scopi diversi:
  - XML progettato per descrivere DATI → cosa sono i dati
  - HTML progettato per visualizzare i dati → come appaiono i dati

Usi di XML

- Separare i dati dal modo con cui vengono visualizzati
- Scambiare i dati tra sistemi eterogenei
- Scambiare informazioni in sistemi B2B
- Condividere dati
- Memorizzare dati
- Supportare nei moderni browser visualizzazione, validazione, embedding in documenti HTML, trasformazioni con XSL, visualizzazioni con CSS
- È un documento di testo → il software che tratta documenti testuali tratta anche XML

Sintassi XML

(http://www.w3schools.com/xml)

- Un documento XML ben formato deve:
  - Cominciare con una direttiva standard che specifica la versione di XML, ad esempio:

    <?xml version= “ 1.0”?>

  - Il documento può contenere elementi. Ogni elemento può contenere sia testo che altri elementi. Un elemento dotato di contenuto è delimitato da due tag: uno di apertura e uno di chiusura. Il tag di chiusura deve avere lo stesso nome del tag di apertura, con in più il prefisso “/”. I nomi di tag sono case sensitive.

Sintassi XML

(http://www.w3schools.com/xml)

- In un documento XML ben formato:
  - Se un elemento è privo di contenuto, ha un solo tag, che deve terminare con il simbolo "/>".
  - Il documento deve avere un elemento radice che racchiude l'intero contenuto.
  - Se un elemento contiene sotto-elementi, il tag di chiusura di ciascun sottoelemento deve precedere il tag di chiusura del sopra-elemento.
  - I valori degli attributi degli elementi devono essere contenuti tra virgolette (singole o doppie).

Elementi

```
<prodotto codice="123kl14">
   <descrizione>
         Forno a ventilazione forzata
   </descrizione>
   <prezzo valuta="USD"> 624 </prezzo>
   <immagine src="foto1.jpg" />
 </prodotto>
```

- Possono avere un contenuto (testo o altri elementi)
- Sono in relazione gerarchica (padre-figlio)
- Possono avere attributi

Attributi

```
<prodotto codice="123">
    <descrizione> Forno </descrizione>
    <prezzo> 104 </prezzo>
 </prodotto>
```

- Proprieta con valori di tipo stringa
- I valori vanno racchiusi tra " "
- Differiscono dagli elementi perché non possono contenere elementi figli
- In genere sono usati per memorizzare metadati, ad esempio l'identificatore di un elemento, la data di creazione o modifica di un dato, ecc..

Namespace

Metodo per evitare

conflitti di nome

```
<table>
 <tr>...</tr>
</table>
<table>
 <product>..</product>
</table>
```

Si introduce un prefisso

```
<h:table xmlns:h="http://www.w3.org/TR/html4/>
   <h:tr>...</h:tr>
</h:table>
<my:table xmlns:my="http://www.me.com/furniture>
   <my:product> ... </my:product>
</my:table>
```

Un XML namespace è una collezione di nomi (attributi, elementi,…), identificata da un URI (Uniform Resource Identifier)

URI = nozione estesa di URL (Uniform Resource Locator)

Un documento XML può far riferimento ad oggetti diversi (DTD, documenti XML, etc.).

È più importante identificare un oggetto che localizzarlo

Document Type Definition (DTD)

- Detta il tipo di un documento, cioè:
  - i tag ammessi
  - le regole di annidamento dei tag
- Scopi:
  - Accordarsi su formato/struttura dei documenti
  - Validare documenti XML secondo certe regole
- Esempio di dichiarazione di un elemento:

- <!ELEMENT PRODOTTO
  (DESCRIZIONE, PREZZO)>
- L'elemento prodotto contiene al suo interno un elemento descrizione seguito da un elemento prezzo

Validazione di un documento XML

- Un documento XML la cui sintassi è corretta (cioè soddisfa le regole di scrittura di documenti XML) è detto "well-formed" (ben formato)

- Un documento che rispetta i vincoli dettati da un DTD è detto "valid" (valido)

Riferimenti

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

XSL

XSL

- Standard W3C: 16 novembre 1999 (1.0)
- XSL = eXtensible Stylesheet Language
- Un foglio di stile è un file in cui sono condensate le specifiche modalità di presentazione
- si può quindi separare (più o meno nettamente) la definizione dei contenuti dalla loro resa grafica

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

XSL

- 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

XSLT

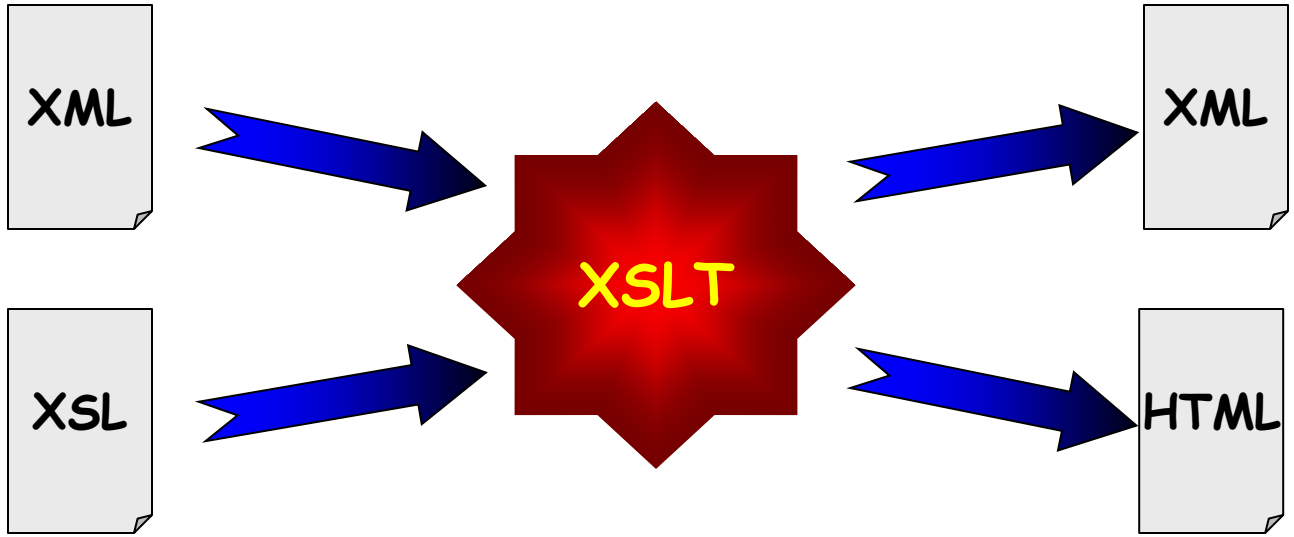
XSL = XSLT + XSL FO

- 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



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

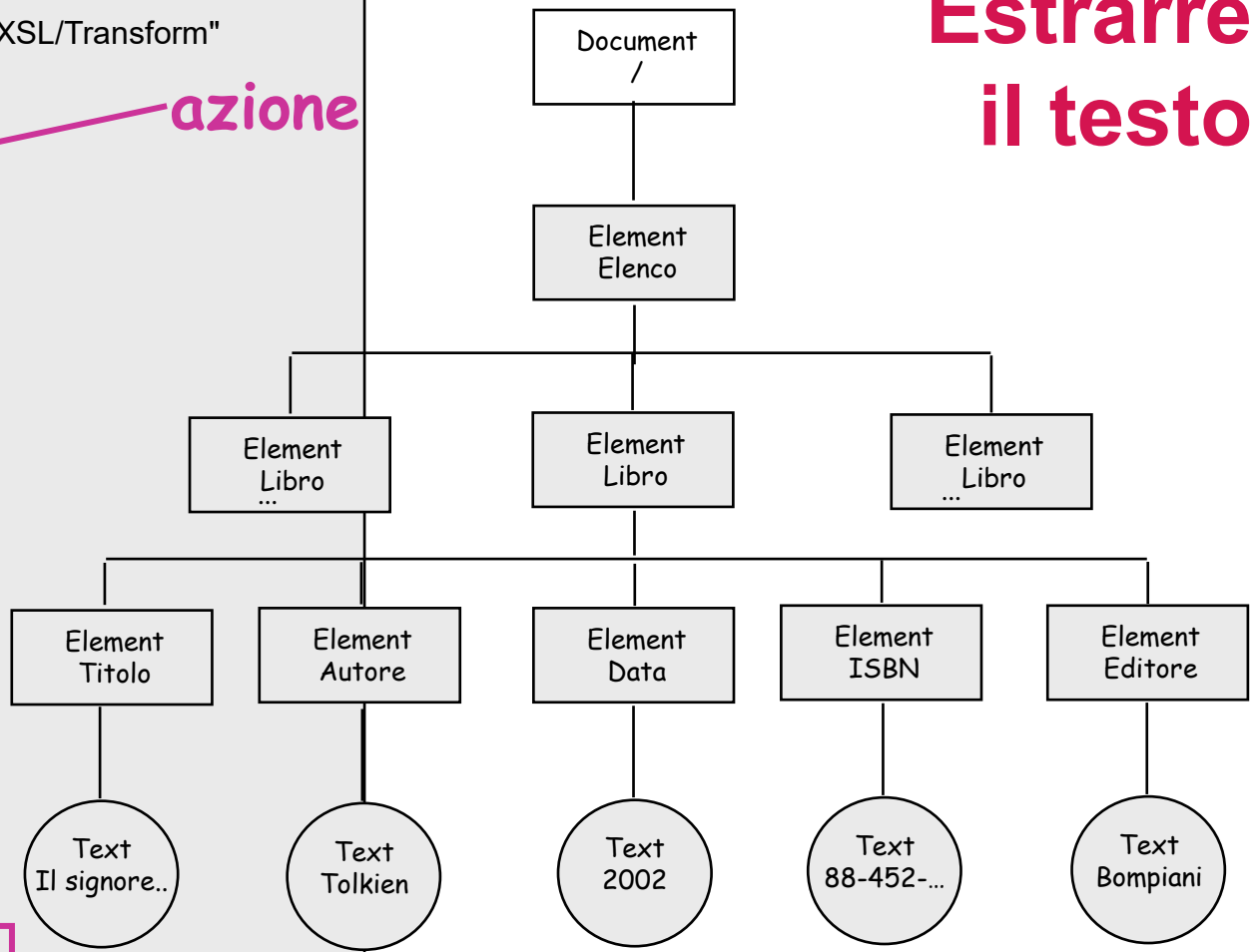
- Il foglio di stile può essere eseguito da un processore XSLT che scandisce il documento XML sorgente, identifica gli elementi per i quali è stato definito un template nel foglio di stile, ed effettua le azioni specificate nel template.

```
<?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>
```

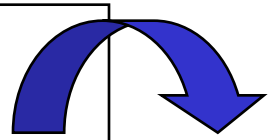
azione

azione

Estrarre il testo



```
<?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>
```



Trasformazione

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

```
<?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>
```

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

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>
```

Linguaggi di interrogazione per XML

Interrogazione di dati XML

- XML è un formato di rappresentazione di dati semi-strutturati
- Un insieme di documenti o anche un singolo documento (ad es: la Divina Commedia) possono essere considerati come depositi interrogabili di informazione
- Nasce l'esigenza di tecnologie di memorizzazione persistente e di interrogazione di dati in formato XML

XQuery

- Linguaggio “alla SQL” per l’interrogazione di dati XML, definito da W3C
  - Inizio lavori 1998
  - Proposta W3C: 15 Febbraio 2001
  - Si basa su XPath per identificare frammenti XML

Esempio di documento

```
<?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>
```

File:

libri.xml

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

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

```
<?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>
```

```
<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>
```

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 - cfr select in OQL)

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:

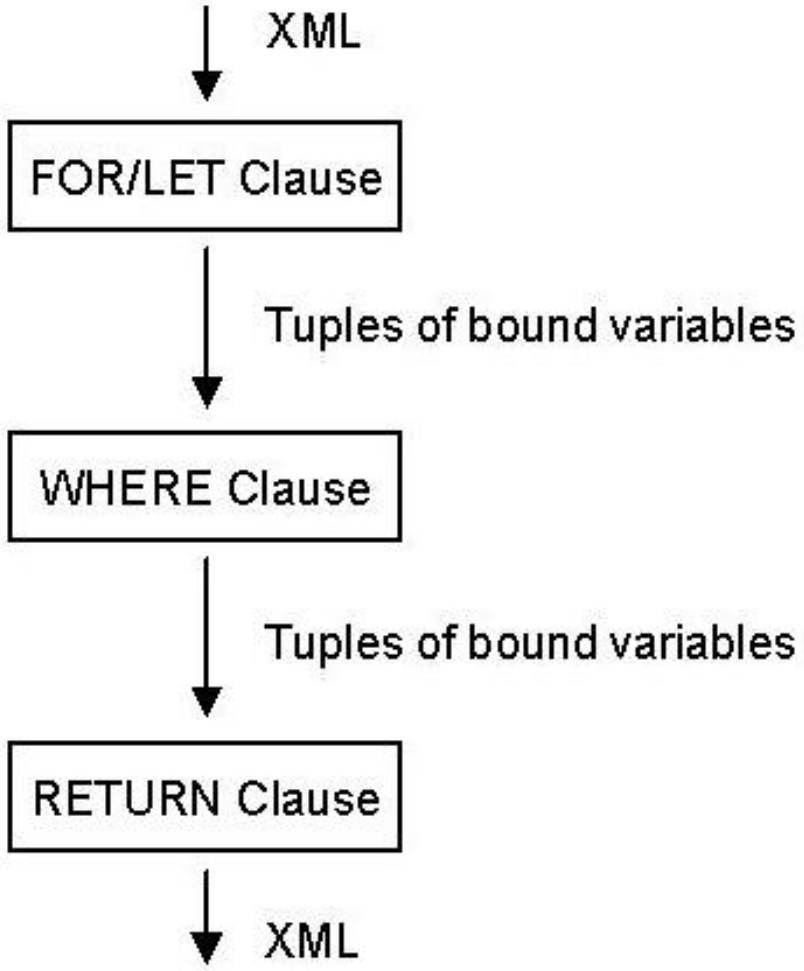
```
for $libro in doc("libri.xml")//Libro
let $a :=  $libro/Autore      (: $a vale l'intera sequenza degli autori del libro :)
return $a
```

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 :=  $libro/Autore
return <libro> { $libro/titolo }
            <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>
```

Path expression che inizia da una sequenza

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

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

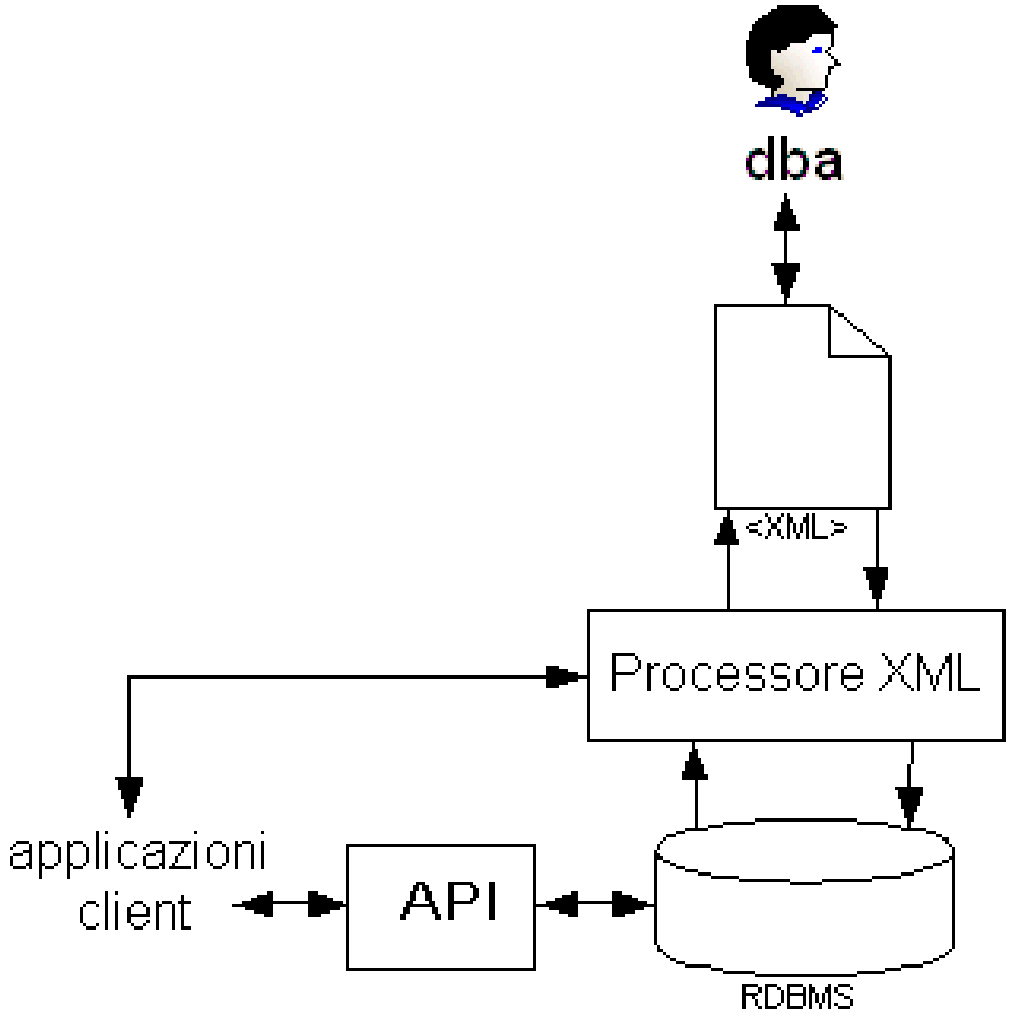
- Memorizzano internamente i documenti XML in tabelle
- Implicano una conversione di ingresso XML→relazionale e in uscita relazionale → XML
- Differiscono per lo schema relazionale usato per mappare i dati XML
  - Fisso, indipendente dal DTD
  - Variabile, dipendente dal DTD
- Normalmente non preservano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

Mapping XML-relazionale

- Custom
  - Il progettista definisce volta per volta a mano lo schema del database in cui riversare documenti XML
- Canonico
  - Insieme di tabelle fisso indipendente dal tipo di documento. Può porre vincoli sullo schema dei documenti XML memorizzabili che devono contenere elementi predefiniti
- Basato sullo schema del documento:
  - Detta regole generali per mappare strutture XML in tabelle, per esempio per mappare un generico elemento e i suoi attributi in una tabella equivalente. Non pone vincoli sui documenti memorizzabili, ma lo schema relazionale risultante varia per diversi tipi di documento

Architettura

- Il DBA | uare query da console
- Le appl | rogrammativa (API) per memorizzare o interroc
- Le conv | ore XML, oppure dell'applicazione (con l'aiuto c



Esempio: supporto per dati XML in Oracle 9i -- query

- Uso di colonne CLOB OracleText (un tipo proprietario) e di operatori che estendono il predicato SQL CONTAINS per il reperimento di elementi in documenti XML
  - Es: INPATH(XPath) restringe una ricerca di stringhe all'interno del frammento XML denotato dall'espressione Xpath

- Uso del tipo SQL 99 XMLType, con metodi per la creazione e manipolazione di dati XML
  - Es: existNode(varchar XPath): conta le occorrenze di sotto-elementi denotati dall'espressione in ingresso; extract(varchar XPath) estrae un frammento di XML

Esempio: supporto per dati XML in Oracle 9i -- API

- XML SQL Utility (XSU) fornisce una API per programmi PL/SQL e Java
  - Trasforma tabelle o view relazionali in documenti XML
  - Divide in frammenti documenti XML e li inserisce in tabelle relazionali
  - Usa un mapping canonico a tre elementi <ROWSET>, <ROW>, <COLUMN_NAME>
  - Supporta tabelle con colonne di tipo complesso
  - Si appoggia su JDBC e facilita la trasformazione del risultato di una query in un albero DOM, la modifica e creazione di dati XML

SQL/XML

- Un'estensione di SQL per sfruttare caratteristiche di XML
- Studiato dal consorzio SQLX
- Parte dello standard ISO
- Contiene varie parti: XMLType, funzioni XSL su colonne di tipo XML, funzioni di generazione di XML da dati relazionali

Generazione di XML da dati relazionali

- Esprimibile mediante funzioni nella clausola SELECT
  - XMLELEMENT: crea una lista di elementi
  - XMLATTRIBUTES: crea attributi all'interno di elementi
  - XMLFOREST: crea una lista di elementi contenenti altri elementi

Esempio

```
CREATE TABLE movie
(
  movie_ID INTEGER,
  title CHAR VARYING (100),
  year_released SMALLINT,
  genre CHAR VARYING (20) ARRAY [ 10 ],
  description CHAR VARYING (300),
  long_description CHAR LARGE OBJECT (2500),
  run_time INTEGER,
  MPAA_rating CHAR (4)
)
```

Costruzione di elementi con attributi

```
SELECT XMLELEMENT( NAME "movieTitle",
            XMLATTRIBUTES(B.year_released as "movieYear")
            B.Title ) AS xmldata
FROM movie AS B
WHERE B.Title like "%Rose%"
```

Risultato

```
<movieTitle movieYear="1979"> The Rose </movieTitle>
<movieTitle movieYear="1986"> The name of the Rose </movieTitle>
<movieTitle movieYear="1989"> The war of the Roses </movieTitle>
```

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