BASI DI DATI 2

BASI DI DATI IN INTERNET

a.a 2024/2025

Prof.ssa G. Tortora

Sommario

- Dati strutturati, semi strutturati e non strutturati.
- Il modello dati gerarchico di XML (ad albero).
- Documenti XML, DTD, e XML Schema.
- Documenti XML e basi di dati.
- Query in XML:
  - Xpath
  - Xquery

Introduzione

- Nonostante l'HTML *(HyperText Markup Language)* è ampiamente usato per la formattazione e la strutturazione dei documenti Web, non è idoneo per specificare dati strutturati che sono estratti dal db.
  - Le applicazioni Internet forniscono interfacce Web per l'accesso ad informazioni memorizzate in uno o più db (**sorgenti di dati**).
- Un nuovo linguaggio XML *(eXtended Markup Language)* sta emergendo come standard per la strutturazione e lo scambio di dati attraverso il Web.
- XML invece di limitarsi a specificare solamente il modo in cui le pagine Web sono formattate per la visualizzazione sullo schermo, può essere usato per fornire informazioni riguardanti la struttura ed il significato dei dati in essi contenuti.
  - I linguaggi per la specifica della struttura dei documenti XML sono chiamati XML DTD *(Document Type Definition)*.
  - Gli aspetti di formattazione sono specificati separatamente usando un linguaggio di formattazione come XSL *(eXtended Stylesheet Language)*.

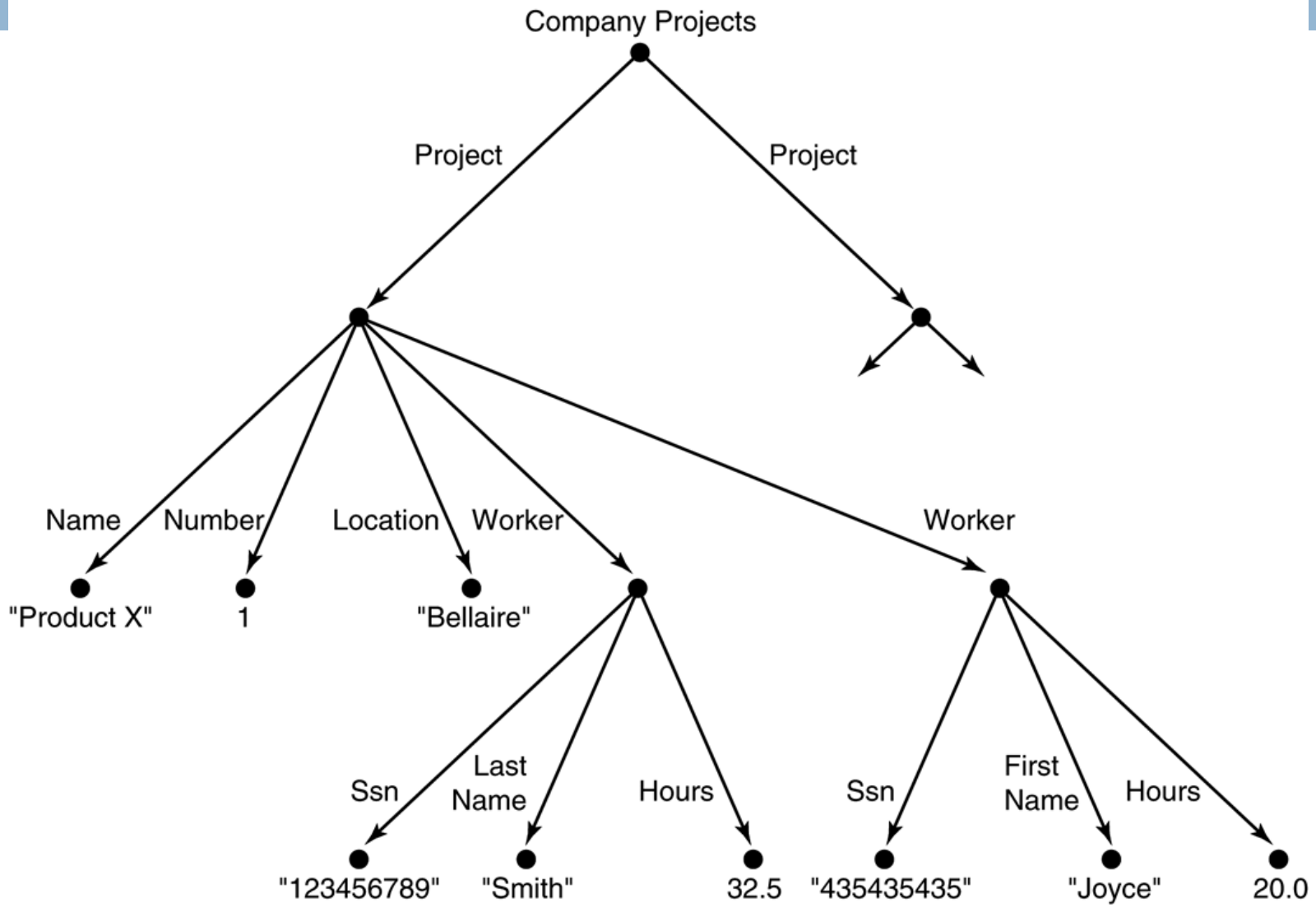
Dati strutturati, semi strutturati e non strutturati

- Le informazioni memorizzate nei db sono conosciute come **dati strutturati** perché sono rappresentate in un formato rigido.
  - Il DMBS si occupa di effettuare i controlli necessari affinché tutti i dati rispettino la struttura e i vincoli specificati nel db.

- In alcune applicazioni i dati sono raccolto con modalità ad-hoc prima ancora di sapere come dovranno essere memorizzati e gestiti.
  - Questi dati possono avere una struttura che però può essere non identica per tutte le informazioni raccolte.
  - Questi tipi di dati sono conosciuti come **dati semi strutturati**.
  - **Es:** i riferimenti bibliografici: libri, report tecnici, pubblicazioni su conferenze o riviste.
  - Nei dati semi strutturati le informazioni sullo schema sono mischiate ai dati, poiché ogni oggetto ha diversi attributi non noti in anticipo.
    - Questi tipi di dati spesso sono detti **dati auto descrittivi**.

- Una terza categoria è conosciuta come **dati non strutturati**.
  - Sono caratterizzati da una presenza molto limitata di informazioni relative ai tipi di dati.
  - **Es:** un documento di testo che contiene informazioni embedded. Le pagine Web in HTML che contengono qualche dato sono considerate come dati non strutturati.

Rappresentazione a grafo di dati semi-strutturati



Dati strutturati, semi strutturati e non strutturati (2)

- I dati semi strutturati possono essere rappresentati come grafi diretti.

- Le **etichette** (o **tag**) sugli archi orientati rappresentano i nomi dello schema:
  - i nomi degli attributi, i tipi degli oggetti (o i tipi delle entità o classi), e le relazioni.
- I nodi interni rappresentano gli oggetti individuali o gli **attributi composti**.
- I nodi foglia rappresentano i valori effettivi degli attributi di tipo semplice (**atomico**).

Documento HTML e dati non strutturati

```
<html>
    <head>
    …
    </head>
    <body>
        <h1>Elenco dei progetti aziendali …</h1>
        <h2>Il progetto Prodotto1:</h2>
        <table width="80%" border="0">
        <tr>
            <td>John Smith</td>
            <td>33 ore settimanali</td>
        </tr>
        <tr>
            <td>James English</td>
            <td>12 ore settimanali</td>
        </tr>
        …
        </table>
    …
    </body>
</html>
```

- HTML utilizza un numero elevato di tag predefiniti che specificano dei comandi per la formattazione dei documenti Web:
  - Tag di delimitazione del documento **<html>…**
  - Intestazione del documento **<head>…**
  - Corpo **<body>…**
  - Stili di intestazione **<h1>….**
  - Tabelle **<table>…<tr>…<td>…**
  - Attributi **width…border…**
  - …

Il modello dati XML

- Un elemento XML complesso chiamato <*progetti*>.

```
<?xml version="1.0" standalone="yes"?>
<projects>

 <project>
  <Name>ProductX</Name>
  <Number>1</Number>
  <Location>Bellaire</Location>
  <DeptNo>5</DeptNo>
  <Worker>
   <SSN>123456789</SSN>
   <LastName>Smith</LastName>
   <hours>32.5</hours>
  </Worker>
  <Worker>
   <SSN>453453453</SSN>
   <FirstName>Joyce</FirstName>
   <hours>20.0</hours>
  </Worker>
 </project>
 </project>
  <Name>ProductY</Name>
  <Number>2</Number>
  <Location>Sugarland</Location>
  <DeptNo >5</DeptNo >
  <Worker>
   <SSN>123456789</SSN>
   <hours>7.5</hours>
  </Worker>
  <Worker>
   <SSN>453453453</SSN>
   <hours>20.0</hours>
  </Worker>
  <Worker>
   <SSN>333445555</SSN>
   <hours>10.0</hours>
  </Worker>
 </project>

...

</projects>
```

Il modello dati XML (ad albero)

- In XML l'oggetto base è il **documento XML**.
- Due concetti fondamentali vengono usati per costruire un documento XML:
  - **elementi** ed **attributi**. In XML gli attributi forniscono informazioni aggiuntive per descrivere gli elementi rispetto l'HTML.
  - Vi sono ulteriori concetti come: *entità*, gli *identificatori*, e i *riferimenti*.
- Come per l'HTML, gli elementi sono identificati dai loro **tag di inizio** e **tag di fine**.
- I nomi dei tag sono racchiusi tra <…>, mentre i tag di fine sono caratterizzati da uno '*slash*' </…>. Gli **elementi complessi** vengono costruiti in modo gerarchico, mentre gli **elementi semplici** contengono solamente valori.
- È facile notare la corrispondenza tra la rappresentazione testuale dell'XML e la struttura ad albero.
- Nella rappresentazione ad albero:
  - i nodi interni rappresentano gli elementi complessi,
  - i nodi foglia rappresentano gli elementi semplici.
  - Per questo motivo il modello XML è chiamato **modello ad albero** o **modello gerarchico**.

Il modello dati XML (ad albero) (2)

- I documenti XML possono essere classificati in tre tipi principali:
  1. ***Documenti XML incentrati sui dati:***
     - Questi documenti possiedono molti dati di dimensioni ridotte che seguono una struttura specifica e possono provenire da un db strutturato.
     - Sono formattati come documenti XML in modo da essere scambiati e visualizzati sul Web.
  2. ***Documenti XML incentrati sul documento*:**
     - Questi sono documenti con grandi quantità di testo (libri ed articoli).
     - Possono esser presenti pochi dati strutturati.
  3. ***Documenti XML ibridi*:**
     - Questi documenti possono possedere parti contenenti dati strutturati e altri costituiti principalmente da testo o da dati non strutturati.

Documenti XML e DTD

- Documento XML ben formato (**Well-Formed):**
  - Deve iniziare con la dichiarazione XML per indicare la versione XML utilizzata ed ogni altro attributo rilevante.
  - Deve seguire le linee-guida sintattiche del modello ad albero.
    - Deve esserci un **unico elemento radice**, ed ogni elemento deve contenere una coppia dei tag di inizio e fine correlati tra loro, contenuta tra i tag dell'**elemento padre**.
  - Un documento XML well-formed è **sintatticamente corretto**.
    - Questo gli permette di essere elaborato da processori generici che attraversano il documento e creano una rappresentazione interna ad albero.
    - **DOM (Document Object Model)**: Permette ai programmi di manipolare la rappresentazione ad albero risultante dall'elaborazione di un documento XML ben formato.
      - L'intero documento deve essere "parsato" completamente per poter essere usato.
    - **SAX**: Permette di processare il documento XML 'al volo' notificando al programma che esegue il processo l'individuazione di ogni tag di inizio e di fine.
- Documento XML **valido**:
  - Un criterio più rigido per un documento XML è la **validità**.
    - In questo caso il documento deve essere well-formed, e in più i nomi degli elementi usati nelle coppie di tag di inizio e di fine devono seguire la struttura specificata in un file **DTD** (**Document Type Definition**) o **XML schema**.

Relazione tra documento e DTD

- Senza DTD :

```
<?XML version="1.0" standalone="yes"?>
<NEWSPAPER>.. </NEWSPAPER>
```

- DTD esterno:

```
<?XML version="1.0" standalone="no"?>
<!DOCTYPE NEWSPAPER SYSTEM "newspaper.dtd">
<NEWSPAPER>.. </NEWSPAPER>
```

- DTD interno:

```
<?XML version="1.0" standalone="no"?>
<!DOCTYPE NEWSPAPER  [
        <!ELEMENT NEWSPAPER(…)>   ...
]>
<NEWSPAPER> … </NEWSPAPER>
```

La notazione XML DTD

- Un * che segue il nome dell'elemento significa che l'elemento può essere ripetuto zero o più volte nel documento.
  - Questo può essere chiamato elemento *a valore multiplo (ripetuto) opzionale*.
- Un + che segue il nome dell'elemento significa che l'elemento può essere ripetuto una o più volte nel documento.
  - Questo può essere chiamato elemento *obbligatorio a valore multiplo (ripetuto)*.
- Un ? che segue il nome dell'elemento significa che l'elemento può essere ripetuto zero o una volta.
  - Questo può essere chiamato elemento a *valore singolo (non ripetuto) opzionale*.

La notazione XML DTD (2)

- Un elemento presente senza **nessuno** dei tre simboli precedenti deve comparire esattamente una volta all'interno del documento.
  - Questo sono elementi *a valore singolo (non ripetuti) necessari*.
- Il **tipo** di un elemento viene specificato tra parentesi di seguito all'elemento stesso.
  - Se le parentesi comprendono nomi di altri elementi, questi ultimi sono i **figli** dell'elemento nella struttura ad albero.
  - Se le parentesi includono la parola chiave **#PCDATA** o uno degli altri tipi di dati disponibili in un DTD XML, l'elemento è un nodo **foglia**.
  - PCDATA (*Parsed Character DATA)* è analogo al tipo di dati stringa.
- Le parentesi (,) possono essere annidate nella specifica degli elementi.
- Un simbolo **barra** (*e1* | *e2*) specifica che nel documento può comparire l'elemento *e1* o *e2.*

Occorrenze di un sottoelemento

- **1 volta**

  <!ELEMENT PRODOTTO (DESCRIZIONE)>

- **1 o più volte**

  <!ELEMENT LISTA (PRODOTTO+)>

- **0 o più volte**

  <!ELEMENT LISTA (PRODOTTO*)>

- **0 o 1 volta**

  <!ELEMENT PRODOTTO (DESCRIZIONE?)>

Modello di contenuto

- Elementi con un elemento figlio:

  <!ELEMENT PRODOTTO (DESCRIZIONE)>

  Es:

  <PRODOTTO> <DESCRIZIONE>…</DESCRIZIONE> </PRODOTTO>

- Elementi con una sequenza di elementi figli:

  <!ELEMENT MAIL (TO, FROM, TITLE, BODY)>

  Es:

  <MAIL> <TO>…</TO> <FROM>…</FROM> <TITLE>…</TITLE>
  <BODY>…</BODY> </MAIL>

- Elementi con contenuto di tipo PCDATA:

  <!ELEMENT DESCRIZIONE (#PCDATA)>

  Es:

  <DESCRIZIONE> Un testo qualsiasi </DESCRIZIONE>

Modello di contenuto (2)

- Contenuto alternativo:
  <!ELEMENT ARTICOLO (TESTO|FOTO)>
  Es:
  <ARTICOLO><TESTO> . . </TESTO></ARTICOLO>
  <ARTICOLO><FOTO> . . </FOTO><ARTICOLO>
- Contenuto misto:
  <!ELEMENT ARTICOLO (#PCDATA|FOTO)*>
  Es:
  <ARTICOLO> testo </ARTICOLO>
  <ARTICOLO><FOTO> . . </FOTO><ARTICOLO>
- Elementi vuoti:
  <!ELEMENT ARTICOLO EMPTY)>
  Es:
  <ARTICOLO/>
- Contenuto arbitrario:
  <!ELEMENT NOTA ANY)>
  Es:
  <NOTA> del testo libero</NOTA>
  <NOTA> <AUTORE>Luca</AUTORE> del testo libero</NOTA>

Dichiarazioni di attributi

- Per ogni elemento dice:
  - quali attributi può avere
  - che valori può assumere ciascun attributo
  - se esiste e qual è il valore di default

- Esempio di dichiarazione di attributo:

```
<!ATTLIST PRODOTTO
    codice      ID                          #REQUIRED
    label       CDATA                       #IMPLIED
    status      (disponibile|terminato) 'disponibile' >
```

Tipi di attributi

- CDATA: dati di tipo carattere
- (val1 | val2 | val3): un valore della lista
- ID: identificatore
- IDREF, IDREFS: valore di un attributo di tipo ID nel documento (o insieme di valori)
- ENTITY, ENTITIES: nome (nomi) di entità
- NMTOKEN, NMTOKENS: caso ristretto di CDATA (una sola parola o insieme di parole)

```
codice    ID                          #REQUIRED
label     CDATA                       #IMPLIED
status    (disponibile|terminato)     'disponibile'
```

DTD: *Esempio*

```
<!ELEMENT ELENCO (PRODOTTO+)>
<!ELEMENT PRODOTTO (DESCRIZIONE, PREZZO?)>
<!ELEMENT DESCRIZIONE (#PCDATA)>
<!ELEMENT PREZZO (#PCDATA)>
```

```
<elenco>
        <prodotto codice="123">
            <descrizione> Forno </descrizione>
            <prezzo> 1040000 </prezzo>
        </prodotto>
        <prodotto codice="432">
            <descrizione> Frigo </descrizione>
        </prodotto>
</elenco>
```

Vincoli sugli attributi

- #REQUIRED: il valore deve essere specificato
- #IMPLIED: il valore può mancare
- #FIXED “valore”: se presente deve coincidere con “valore”
- **Es:**

  <!ELEMENT FAX (..)>

  <!ATTLIST FAX Mittente CDATA **#FIXED “Politecnico di Milano"**>

- Default : si può specificare un valore come default, usato quando l’attributo è mancante:

  <!ELEMENT SCONTO EMPTY>

  <!ATTLIST SCONTO Valore CDATA “10">

  <SCONTO/> → vale 10

  <SCONTO valore=”15”/> → vale 15

DTD: *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>
]>
```

DTD: *Esempio 2*

- Un file DTD chiamato *projects*.

```
<!DOCTYPE projects [
  <!ELEMENT projects (project+)>
  <!ELEMENT project (Name, Number, Location, DeptNo?, Workers)>
  <!ELEMENT Name (#PCDATA)>
  <!ELEMENT Number (#PCDATA)>
  <!ELEMENT Location (#PCDATA)>
  <!ELEMENT DeptNo (#PCDATA)>
  <!ELEMENT Workers (Worker*)>
  <!ELEMENT Worker (SSN, LastName?, FirstName?, hours)>
  <!ELEMENT SSN (#PCDATA)>
  <!ELEMENT LastName (#PCDATA)>
  <!ELEMENT FirstName (#PCDATA)>
  <!ELEMENT hours (#PCDATA)>
] >
```

Limitazioni degli XML DTD

- Le DTD non consentono la definizione di tipi di dati, pertanto non possiamo avere un controllo accurato sul contenuto degli elementi di un documento XML né sul valore degli attributi.

- Le DTD hanno una loro sintassi e richiedono processori particolari.
  - Sarebbe un vantaggio specificare gli schemi dei documenti XML utilizzando le regole sintattiche di XML.
  - In tal caso gli stessi processori per i documenti XML possono processare le descrizioni degli schema XML.

- Tutti gli elementi nella DTD sono sempre obbligati a seguire l'ordine particolare del documento:
  - Non sono permessi elementi non ordinati.

- Un documento XML puó dipendere da un solo DTD che definisce tutta la grammatica applicabile.
  - Non consentono la presenza in un documento XML di tag definiti in schemi diversi.

XML Schema

- **Scopo:**
  - Definire gli elementi e la composizione di un documento XML in modo più efficiente 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

XSD vs DTD

- XML Schema = insieme di elementi XML standard per definire schemi di documenti detti XSD.
- 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).

Namespace XML

- È necessario identificare il particolare insieme di elementi (tag) del linguaggio XML schema usati specificando un file memorizzato in un sito Web.
- La seconda linea nel prossimo esempio specifica il file usato. In particolare:
  - "**http://www.w3.org/2001/XMLSchema**"
- Ogni definizione di questo tipo è detta **namespace XML**.
- Il nome del file è assegnato alla variabile **xs** usando l'attributo xmlns (XML namespace), e questa variabile è usata come prefisso di tutti i comandi di XML schema.

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 elementi o attributi).
- Forme di dichiarazione:

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

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

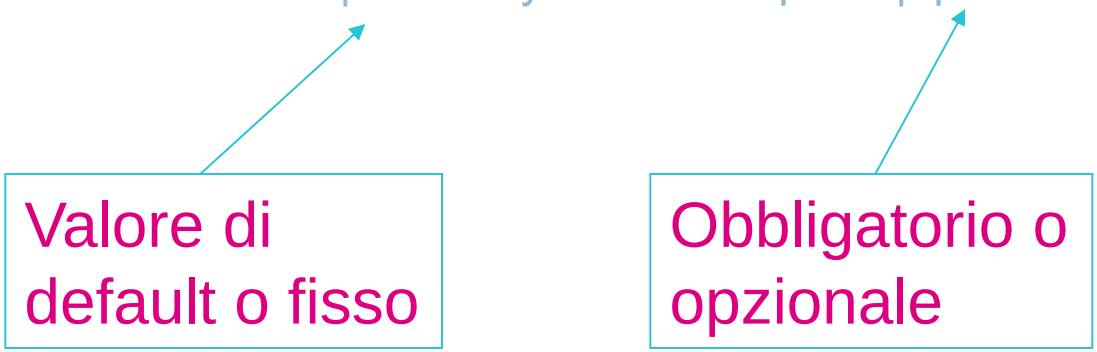
```
<xs:element name="età" type="xs:integer"/>
<xs:element name="cognome" type="xs:string"/>
```

- Esempi di elementi semplici XML:

```
<età> 65 </età>
<cognome> Rossi </cognome>
```

Attributi

- 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>

Restrizioni

- 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="Golf"/>
    <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.

Specifica del contenuto: costrutto 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.

Elementi EMPTY

- Gli elementi senza sottoelementi interni si dichiarano come complex type privi di sottoelementi (element):

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

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.

  - Per gli elementi multivalore si deve impostare: maxOccurs="unbounded"
  - Per i valori nulli si deve impostare: minOccurs = 0

- 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"
       maxOccurs="10" minOccurs="0"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Definizioni di tipi riusabili

- Consente di definire tipi e riusarli per:
  - Tipare attributi.
  - Definire altri tipi.

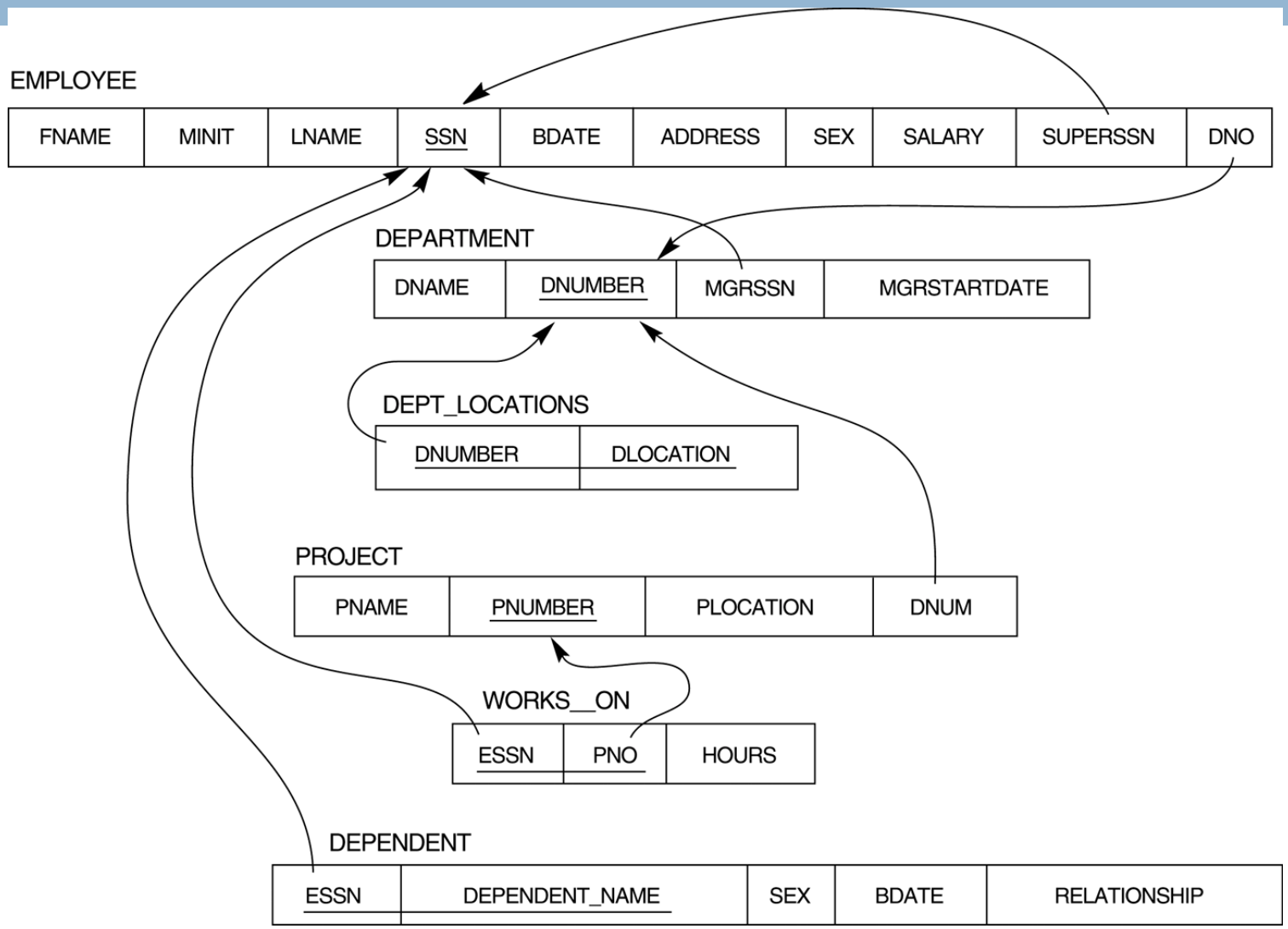
- *ATTENZIONE: Definizione di tipo, non di elemento*

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

- Riuso del tipo, per definire elementi/attributi:

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

II Database Company



La definizione del Database Company in SQL

```
CREATE TABLE EMPLOYEE
        ( FNAME          VARCHAR(15)      NOT NULL ,
          MINIT          CHAR ,
          LNAME          VARCHAR(15)      NOT NULL ,
          SSN            CHAR(9)          NOT NULL ,
          BDATE          DATE
          ADDRESS        VARCHAR(30) ,
          SEX            CHAR ,
          SALARY         DECIMAL(10,2) ,
          SUPERSSN       CHAR(9) ,
          DNO            INT              NOT NULL ,
    PRIMARY KEY (SSN) ,
    FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN) ,
    FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER) ) ;
CREATE TABLE DEPARTMENT
        ( DNAME          VARCHAR(15)      NOT NULL ,
          DNUMBER        INT              NOT NULL ,
          MGRSSN         CHAR(9)          NOT NULL ,
          MGRSTARTDATE   DATE ,
        PRIMARY KEY (DNUMBER) ,
        UNIQUE (DNAME) ,
        FOREIGN KEY (MGRSSN) REFERENCES EMPLOYEE(SSN) ) ;
CREATE TABLE DEPT_LOCATIONS
        ( DNUMBER        INT              NOT NULL ,
          DLOCATION      VARCHAR(15)      NOT NULL ,
        PRIMARY KEY (DNUMBER, DLOCATION) ,
        FOREIGN KEY (DNUMBER) REFERENCES DEPARTMENT(DNUMBER) ) ;
```

```
CREATE TABLE PROJECT
        ( PNAME          VARCHAR(15)      NOT NULL ,
          PNUMBER        INT              NOT NULL ,
          PLOCATION      VARCHAR(15) ,
          DNUM           INT              NOT NULL ,
        PRIMARY KEY (PNUMBER) ,
        UNIQUE (PNAME) ,
        FOREIGN KEY (DNUM) REFERENCES DEPARTMENT(DNUMBER) ) ;
CREATE TABLE WORKS_ON
        ( ESSN           CHAR(9)          NOT NULL ,
          PNO            INT              NOT NULL ,
          HOURS          DECIMAL(3,1)     NOT NULL ,
        PRIMARY KEY (ESSN, PNO) ,
        FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN) ,
        FOREIGN KEY (PNO) REFERENCES PROJECT(PNUMBER) ) ;
CREATE TABLE DEPENDENT
        ( ESSN           CHAR(9)          NOT NULL ,
          DEPENDENT_NAME VARCHAR(15)      NOT NULL ,
          SEX            CHAR ,
          BDATE          DATE ,
          RELATIONSHIP   VARCHAR(8) ,
            PRIMARY KEY (ESSN, DEPENDENT_NAME) ,
            FOREIGN KEY (ESSN) REFERENCES EMPLOYEE(SSN) ) ;
```

Gli statement SQL2 per definire lo schema del database *"Company"*

XML Schema (Company): *Esempio*

- Un documento XML Schema *company*.

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
      <xsd:documentation xml:lang="en">Company Schema (Element Approach) -
        Prepared by Babak Hojabri</xsd:documentation>
  </xsd:annotation>
<xsd:element name="company">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="department" type="Department" minOccurs="0"
            maxOccurs="unbounded" />
        <xsd:element name="employee" type="Employee" minOccurs="0"
            maxOccurs="unbounded">
          <xsd:unique name="dependentNameUnique">
            <xsd:selector xpath="employeeDependent" />
            <xsd:field xpath="dependentName" />
          </xsd:unique>
        </xsd:element>
        <xsd:element name="project" type="Project" minOccurs="0"
            maxOccurs="unbounded" />
      </xsd:sequence>
    </xsd:complexType>
```

...

XML Schema: *Esempio (2)*

...

```
<xsd:unique name="departmentNameUnique">
    <xsd:selector xpath="department" />
    <xsd:field xpath="departmentName" />
</xsd:unique>
<xsd:unique name="projectNameUnique">
    <xsd:selector xpath="project" />
    <xsd:field xpath="projectName" />
</xsd:unique>
<xsd:key name="projectNumberKey">
    <xsd:selector xpath="project" />
    <xsd:field xpath="projectNumber" />
</xsd:key>
<xsd:key name="departmentNumberKey">
    <xsd:selector xpath="department" />
    <xsd:field xpath="departmentNumber" />
</xsd:key>
<xsd:key name="employeeSSNKey">
    <xsd:selector xpath="employee" />
    <xsd:field xpath="employeeSSN" />
</xsd:key>
<xsd:keyref name="departmentManagerSSNKeyRef" refer="employeeSSNKey">
    <xsd:selector xpath="department" />
    <xsd:field xpath="departmentManagerSSN" />
</xsd:keyref>
<xsd:keyref name="employeeDepartmentNumberKeyRef"
    refer="departmentNumberKey">
    <xsd:selector xpath="employee" />
    <xsd:field xpath="employeeDepartmentNumber" />
</xsd:keyref>
<xsd:keyref name="employeeSupervisorSSNKeyRef" refer="employeeSSNKey">
    <xsd:selector xpath="employee" />
    <xsd:field xpath="employeeSupervisorSSN" />
</xsd:keyref>
<xsd:keyref name="projectDepartmentNumberKeyRef"
    refer="departmentNumberKey">
    <xsd:selector xpath="project" />
    <xsd:field xpath="projectDepartmentNumber" />
</xsd:keyref>
<xsd:keyref name="projectWorkerSSNKeyRef" refer="employeeSSNKey">
    <xsd:selector xpath="project/projectWorker" />
    <xsd:field xpath="SSN" />
</xsd:keyref>
<xsd:keyref name="employeeWorksOnProjectNumberKeyRef"
    refer="projectNumberKey">
    <xsd:selector xpath="employee/employeeWorksOn" />
    <xsd:field xpath="projectNumber" />
</xsd:keyref>
</xsd:element>
```

XML Schema: *Esempio* (3)

```
<xsd:complexType name="Department">
  <xsd:sequence>
    <xsd:element name="departmentName" type="xsd:string" />
    <xsd:element name="departmentNumber" type="xsd:string" />
    <xsd:element name="departmentManagerSSN" type="xsd:string" />
    <xsd:element name="departmentManagerStartDate" type="xsd:date" />
    <xsd:element name="departmentLocation" type="xsd:string"
        minOccurs="0" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Employee">
  <xsd:sequence>
    <xsd:element name="employeeName" type="Name" />
    <xsd:element name="employeeSSN" type="xsd:string" />
    <xsd:element name="employeeSex" type="xsd:string" />
    <xsd:element name="employeeSalary" type="xsd:unsignedInt" />
    <xsd:element name="employeeBirthDate" type="xsd:date" />
    <xsd:element name="employeeDepartmentNumber" type="xsd:string" />
    <xsd:element name="employeeSupervisorSSN" type="xsd:string" />
    <xsd:element name="employeeAddress" type="Address" />
    <xsd:element name="employeeWorksOn" type="WorksOn" minOccurs="1"
        maxOccurs="unbounded" />
    <xsd:element name="employeeDependent" type="Dependent" minOccurs="0"
        maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Project">
  <xsd:sequence>
    <xsd:element name="projectName" type="xsd:string" />
    <xsd:element name="projectNumber" type="xsd:string" />
    <xsd:element name="projectLocation" type="xsd:string" />
    <xsd:element name="projectDepartmentNumber" type="xsd:string" />
    <xsd:element name="projectWorker" type="Worker" minOccurs="1"
        maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Dependent">
  <xsd:sequence>
    <xsd:element name="dependentName" type="xsd:string" />
    <xsd:element name="dependentSex" type="xsd:string" />
    <xsd:element name="dependentBirthDate" type="xsd:date" />
    <xsd:element name="dependentRelationship" type="xsd:string" />
  </xsd:sequence>
</xsd:complexType>
<xsd:complexType name="Address">
  <xsd:sequence>
    <xsd:element name="number" type="xsd:string" />
    <xsd:element name="street" type="xsd:string" />
    <xsd:element name="city" type="xsd:string" />
    <xsd:element name="state" type="xsd:string" />
  </xsd:sequence>
```

...

XML Schema: *Esempio* (4)

```
    </xsd:complexType>
    <xsd:complexType name="Name">
       <xsd:sequence>
          <xsd:element name="firstName" type="xsd:string" />
          <xsd:element name="middleName" type="xsd:string" />
          <xsd:element name="lastName" type="xsd:string" />
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="Worker">
       <xsd:sequence>
          <xsd:element name="SSN" type="xsd:string" />
          <xsd:element name="hours" type="xsd:float" />
       </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="WorksOn">
       <xsd:sequence>
          <xsd:element name="projectNumber" type="xsd:string" />
          <xsd:element name="hours" type="xsd:float" />
       </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

Annotations and documentation

- Gli elementi **xsd:annotation** e **xsd:documentation** sono utilizzati per fornire commenti ed altre descrizioni nel documento XML.

- L'attributo **xml:lang** dell'elemento **xsd:documentation** specifica il linguaggio usato ("en" specifica la lingua inglese).

Specifica delle chiavi

- Per specificare una chiave primaria è utilizzato il tag **xsd:key**.
- Per specificare le chiavi esterne è usato il tag **xsd:keyref**.
  - Quando si specifica una chiave esterna, l'attributo **refer** del tag **xsd:keyref** indica la chiave primaria riferita mentre i tag **xsd:selector** and **xsd:field** specificano il tipo dell'elemento riferito e la chiave esterna.

Interrogazioni in XML

- Tra le varie proposte avanzate come linguaggi di interrogazione XML sono emersi due standard:
  1. **XPath** che fornisce dei costruttori di linguaggio per la specifica di ***path expression*** volti ad identificare determinati nodi (elementi) che corrispondono a particolari pattern contenuti nel documento XML.
  2. **XQuery** è un linguaggio di interrogazione più generale che utilizza le espressioni XPath ma che possiede costrutti aggiuntivi.

XPath

- Una espressione XPath ritorna una collezione di nodi (elementi) che soddisfa determinati pattern in essa specificati.

- I nomi nella espressione XPath sono i nomi dei nodi contenuti nell'albero del documento XML, che possono essere i nomi di tag (elementi) o nomi di attributi, possibilmente con l'aggiunta di **condizioni di qualifica** volte a limitare ulteriormente i nodi che soddisfano i pattern.
- Nella specifica di un path sono utilizzati due separatori:

  lo slash (/) singolo ed il doppio slash (//):
  - Lo slash singolo prima di un tag specifica che quel tag deve essere un figlio diretto del tag precedente (padre),
  - mentre il doppio slash indica che il tag può essere un discendente di qualunque livello.

- È consuetudine includere il nome del file in ogni interrogazione XPath in modo da permettere di specificare qualsiasi nome di file o nome di percorso che specifica un file presente nel Web.
- **Es:**

  se **www.company.com/info.XML** => il documento XML COMPANY

  allora **doc(www.company.com/info.XML)/company** =>

  espressione XPath

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)

Esempio di documento 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>
```

File: libri.xml

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.
- **Es:**

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

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

Esempi di path expression

```
<?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

- **Es:**

  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

- **Es:**

  doc(“libri.xml”)//Autore

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

Risultato:

<Autore>J.R.R. Tolkien</Autore>

<Autore>Umberto Eco</Autore>

<Autore>F. Dürrenmatt</Autore>

Condizione sulla posizione dei sottoelementi e uso di wildcard

- **Es:**

  doc (“libri.xml”)/Elenco/Libro[2]/*

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

Risultato:

<Titolo>Il nome della rosa</Titolo>

<Autore>Umberto Eco</Autore>

<Data>1987</Data>

<ISBN>55-344-2345-1</ISBN>

<Editore>Bompiani</Editore>

Esempi XPath sul documento XML schema di company

1. Ritorna il nodo radice di COMPANY e tutti I suoi nodi discendenti (questo significa che ritorna l'intero documento XML).
2. Ritorna tutti i nodi (elementi) department ed il loro sottoalbero.
3. Ritorna tutti i nodi employeeName che sono figli diretti del nodo employee, così che quel nodo employee ha un altro figlio employeeSalary il cui valore è maggiore di 70000.
4. Ritorna lo stesso risultato di (4) eccetto che venga specificato l'ntero nome del path.
5. Questo ritorna tutti nodi projectWorker ed i loro nodi che sono figli del path ***/company/project*** e che hanno un nodo figlio hours con valore maggiore o uguale a 20.0.

Le espressioni sono nella prossima slide.

Esempi XPath sul documento XML schema di company (2)

1. `/company`
2. `/company/department`
3. `//employee [employeeSalary gt 70000]/employeeName`
4. `/company/employee [employeeSalary gt 70000]/employeeName`
5. `/company/project/projectWorker [hours ge 20.0]`

XQuery

- XQuery usa le espressioni XPath ma ha ulteriori costrutti.
  - Si basa su XPath per identificare frammenti XML:
    - È basato sulla elaborazione di **sequenze di nodi**.
  - Una interrogazione XQuery è un'espressione complessa che consente di **estrarre parti di un documento e costruire un altro documento.**

- XQuery permette di specificare interrogazioni più generali su uno o più documenti XML.

- La forma tipica di una interrogazione XQuery è conosciuta come **espressione FLWR**,
- FLWR indica le quattro principali clausole di XQuery ed hanno il seguente formato:

FOR < *itera i valori delle variabili su sequenze di nodi* >
LET < *variabili legate a collezioni di nodi* >
WHERE < esprime *condizioni di qualificazione sui legami* >
RETURN < *specificazione del risultato dell'interrogazione* >

Le espressioni FLWOR

- In una singola Xquery :
  - zero o più istanze della clausola **FOR**;
  - zero o più istanze della clausola **LET**;
  - al più una istanza di ciascuna delle clausole **WHERE** e **ORDER** (opzionali);
  - esattamente una istanza della clausola **RETURN**.
- Le variabili sono prefissate da $; la clausola LET assegna una variabile a una particolare espressione per il resto della query;
- La clausola FOR assegna una variabile a variare su ciascuno degli elementi individuali di una sequenza;
- La clausola WHERE specifica condizioni addizionali sulla selezione;
- La clausola ORDER su quale item gli elementi risultanti dovranno essere ordinati;
- La clausola RETURN specifica quali elementi o attributi dovranno essere ritrovati dagli items che soddisfano le condizioni della query.

Espressioni FOR

- **Es:**

  ```
  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
```

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.
- **Es:**

```
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 RETURN

- Genera l'output di un'espressione FLWR che può essere:
  - Un nodo
  - Un "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.
- **Es:**

```
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

- **Es (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>

Ordinare il risultato

- **Es:**

```
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 di aggregazione

- **Es:**

```
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 !!!

XQuery sul documento XML schema di company: Esempio 1

1. FOR $x IN
   doc(www.company.com/info.xml)
   //employee [employeeSalary gt 70000]/employeeName
   RETURN <res> $x/firstName, $x/lastName </res>

- Questa interrogazione ritrova il nome ed il cognome degli impiegati che guadagnano più di 70000. La variabile $x è legata ad ogni elemento employeeName figlio degli elementi employee per cui il valore di employeeSalary è maggiore di 70000.

XQuery sul documento XML schema di company: Esempio 2

2. FOR $x IN
   doc(www.company.com/info.xml)/company/employee
   WHERE $x/employeeSalary gt 70000
   RETURN <res> $x/employeeName/firstName, $x/employeeName/lastName </res>

- Questo è un modo alternativo per ritrovare gli stessi elementi della query (1).

XQuery sul documento XML schema di company: Esempio 3

3. FOR $x IN
doc(www.company.com/info.xml)/company/project[projectNumber=5]/projectWorker,
$y IN doc(www.company.com/info.xml)/company/employee
WHERE $x/hours gt 20.0 AND $y.ssn=$x.ssn
RETURN <res> $y/employeeName/firstName, $y/employeeName/lastName, $x/hours </res>

- Questa query illustra come un'operazione di **join** può essere eseguita usando più di una variabile. La variabile $x è legata a ogni elemento projectWorker che è figlio del progetto numero 5, mentre la variabile $y è legata ad ogni elemento employee. La condizione di join confronta i valori ssn al fine di recuperare i nomi degli impiegati.