A.A. 2020/21

BASI DI DATI - XML

SARA MIGLIORINI

FABS :)

NOTA

Questi appunti/sbobinatura/versione “discorsiva” delle slides sono per mia utilità personale, quindi pur avendole revisionate potrebbero essere ancora presenti typos, commenti/aggiunte personali (che anzi, lascio di proposito) e nel caso peggiore qualche inesattezza!

Comunque spero siano utili! 

Questa sbobina fa parte della mia collezione di sbobinature,
che è disponibile (e modificabile!) insieme ad altre in questa repo:
https://github.com/fabfabretti/sbobinamento-seriale-uniVR

Contents

Tecnologie alternative: XML

È un linguaggio di marcatura proposto dal W3C, che definisce una sintassi generica per contrassegnare i dati di un documento elettronico con tag semplici e leggibili. Si può usare in contesti molto diversi:

- Pagine web
- Scambio di dati tra applicazioni web
- Grafica vettoriale
- Cataloghi di prodotti
- Sistiemi di gestione di messaggi vocali (???)
- …

Nasce, in realtà, come evoluzione di altri linguaggi di marcatura:

- 1986→ **SGML**: Standard Generalized Markup Language
  - Linguaggio di marcatura strutturato per la rappresentazione eletteronica di documenti di testo
- 1995 → **HTML**: HyperText Markup Language
  - Applicazione di SGML che permette di descrivere come il contenuto di un documento verrà presentato in un'interfaccia
- 1998 → **XML**: eXtensible Markup Language
  - Versione "leggera" di SGML che consente una formattazione semplice e molto flessibile.

HTML	XML
• Insieme fisso di tag • Descrizione degli aspetti di presentazione del documento • Usato solo per la costruzione di pagine web	• Insieme non fisso di tag: si possono personalizzare • Desrizione del contenuto informativo del documento • Usato in molti domini diversi.

```
<h1>
    Essential XML: <br>
        Oltre il Markup
</h1>

<ul>
    <li>
        Don Box
    </li>
    <li>
        Aaron Skonnard
    </li>
    <li>
        John Lam
    </li>

</ul>

<i>
    Addison-Wesley
</i>
```

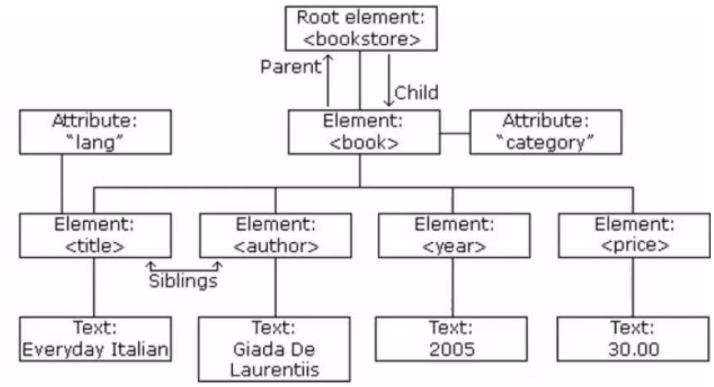
```
<titolo>
    Essential XML:
        Oltre il Markup
</titolo>

<autore>
    Don Box
</autore>
<autore>
    Aaron Skonnard
</autore>
<autore>
    John Lam
</autore>
<casa editrice>
    Addison-Wesley
</casa editrice>
```

Sintassi

XML è **case sensitive**. Ciascun elemento è caratterizzato da:

- **Tag iniziale** → <nome_elemento>
- **Tag finale** → </nome_elemento>
- **Contenuto**
  → può essere un valore atomico o un valore strutturato attraverso altri XML (elementi figli).
  - Esistono anche elementi vuoti.
  - Il contenuto di un elemento può essere anche misto: posso trovare contemporaneamente valori e altri tag.
  - Un elemento può anche essere corredato di attributi, ovvero coppie nome-valore. Ad esempio <persona cod_fisc = "CIAO">
- **Struttura variabile**: un elemento può essere opzionale

Può essere rappresentato da una struttura ad albero, dove ciascun nodo è un tag e i nodi foglia sono i valori.



Documento ben formato

XML è più restrittivo di HTML nella sintassi. Un documento è ben formato se:

- **Ha una sola radice**
- **L'annidamento dei tag è corretto**
- **Tutti i tag sono aperti e chiusi**
- **I valori degli attributi sono specificati fra virgolette**

Elementi contro attributi

Attributi	Elementi
<person sex="female"> <firstname>Anna</firstname> <lastname>Smith</lastname> </person>	<person> <sex>female</sex> <firstname>Anna</firstname> <lastname>Smith</lastname> </person>
→ Non può contenere più valori → Non può contenere info ulteriormente strutturate → Non possono essere estesi facilmente → Sono difficili da leggere	→ Rappresentare un'informazione come sottoelemento ci dà **più struttura** e più flessibilità

Insomma, meglio usare gli elementi per le informazioni e gli attributi per la meta-informazione.

Sintassi dei nomi

- Possono essere costituiti da **qualunque carattere alfanumerico**
- Ammessi **underscore_, trattino -**, **punto .**
- Possono iniziare solo con lettere, ideogrammi o con il carattere underscore.

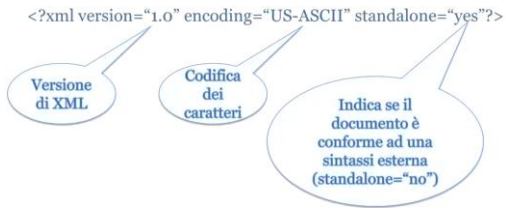
Dati a struttura variabile

```
<messages>
    <note id="501">
      <to>Tove</to>
      <from>Jani</from>
      <body>Don't forget me this
       weekend!</body>
    </note>
    <note id="502">
      <to>Jani</to>
      <from>Tove</from>
      <heading>Re: Reminder</heading>
      <body>I will not</body>
    </note>
    <note id="505">
      <to>Jani</to><from>Tove</from>
      <body>I will not</body>
      <cc>John</cc>
    </note>
</messages>
```

L'elemento note cambia struttura ogni volta:

- Nel primo caso abbiamo solo to, from, body
- Nel secondo caso c'è to, from, heading e body
- Nel terzo abbiamo to, from, body, cc.

Intestazione



Ogni documento XML inizia con una dichiarazione XML.

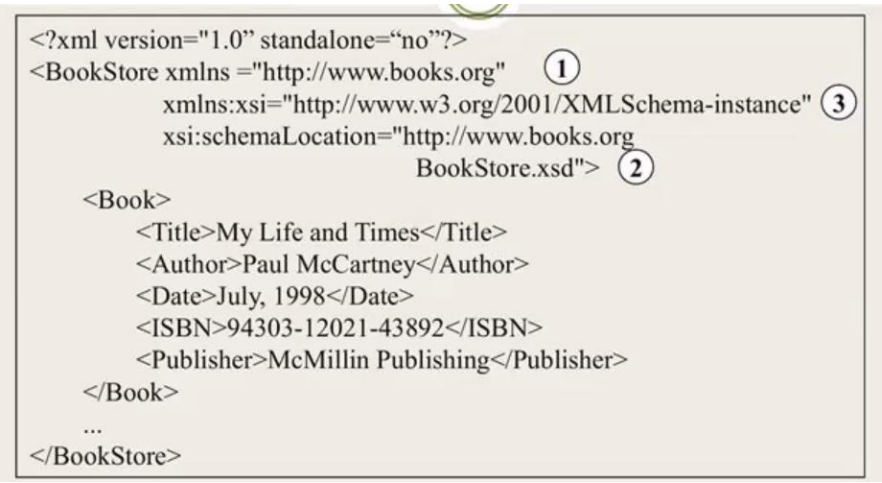
Validazione dei documenti XML e file XSD

Possiamo descrivere un documento XML che descriva la sintassi di un altro XML; per esempio quali sono i tag ammessi, quale sia la struttura dei tag e così via.

La sintassi di un file XML si scrive attraverso uno schema di documento in XMLSchema, ovvero un file .XSD. Ogni documento XML è valido se è ben formato e rispetta la sintassi specificata nel suo file XSD.

Referenziare uno schema nell'XML

xmlns = ".."	→ specifica la dichiarazione di default di un namespace, ovvero dice al validatore che qualunque elemento senz prefisso viene da quel namespace.
xmlns:xsi = ".."	→ ho aggiunto un altro namespace dandogli xsi come prefisso; ogni volta che mi ci riferisco devo usare xsi.
xsi:schemaLocation=".."	→ dice al validator che il namespace è definito dal file BookStore.xsd



```
<?xml version="1.0" standalone="no"?>
<BookStore xmlns ="http://www.books.org"                    (1)
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  (3)
           xsi:schemaLocation="http://www.books.org
                               BookStore.xsd">              (2)
    <Book>
        <Title>My Life and Times</Title>
        <Author>Paul McCartney</Author>
        <Date>July, 1998</Date>
        <ISBN>94303-12021-43892</ISBN>
        <Publisher>McMillin Publishing</Publisher>
    </Book>
    ...
</BookStore>
```

XSD: XML Schema

Parentesi storica: i DTD

Prima di definire l'XMLSchema esisteva il DTD – document type definition, che specificava:

I DTD sono stati male accolti per diversi motivi:

DTD	XSD
• Sintassi diversa da quella dell'XML → inconsistente • Capability nei datatype molto limitata › Per esempio non puoi dire "<elevation> deve essere in integer da 0 a 12000" › DTD ha 10 datatypes, XML 44+	• Migliori datatypes › 44 di default › Custom: ad esempio "basato su stringa ma gli elementi devono essere 'dd-dddd' dove d è digit • Stessa sintassi dei documenti istanza • Circa object-oriented (puoi estendere oggetti) • Esprime insiemi, aka permette che i figli siano dichiarati in qualunque ordine • Può esprimere che un elemento debba essere unico • Definisce contenuto nullo • Definisce elementi sostitutibili (tipo pubblicazione anziché libro)

Caratteristiche

Un XMLSchema è un vocabolario XML per esprimere le regole del nostro data buisness.

XML ci permette di **esprimere constraint**. Per verificare che i dati siano accettabili esistono tools detti XML Schema validators e fanno questa verifica.

Mentre l'XML ha un insieme di tag completamente liberi, XMLSchema abbiamo un vocabolario ben preciso che permette di rappresentare solo elementi prefissati, tramite i quali definisco una nuova sintassi.

Considerando un grosso programma, una parte del nostro progetto specifica il lavoro vero e proprio, e un'altr aparte che verifica la struttura e il contenuto dei dati. Si stima che questa seconda parte sia il 60%; quindi, se i nostri dati hanno una struttura rappresentabile tramite XMLSchema, abbiamo validatori automatici che ci salvano quell'enorme fetta di lavoro.

Quindi, l'XML-schema è tre cose assieme:

- **Modello dei dati**: con l'XML schema possiamo specificare come saranno organizzati i dati in XML e i datatypes.
- **Contratto**: nel momento in cui specifico l'XSD e conveniamo con il consumatore che i miei dati sono in questo formato posso validarli.
- **Sorgente di metadati**: troviamo all'interno moltissime informazioni sui dati quali il loro tipo, i valori ammissibili, come sono relazionati fra loro.

Non ha limiti: ad esempio, tramite XML si possono rappresentare GUI che potranno essere verificate automaticamente. Inoltre, gli smart-editor permettono di dare suggerimenti sul codice runtime basandosi su XML.

Dal punto di vista della pura rappresentazione dei dati esistono delle varianti agli XML come il formato **JSON**, che rappresenta dati semistrutturati ma è tipicamente meno efficace per alcune applicazioni.

Namespaces

Un namespace è una **URI**, Uniform Resource Identifier, che specifica **dove si trova la sintassi di quell'insieme di elementi.**

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
        targetNamespace=http://www.books.org
        xmlns=http://www.books.org
        elementFormDefault="qualified">
```

targetNamespace = link	Specifica che gli elementi specificati in questo schema vanno al link specificato.

xmlns:prefisso = "URI"
 	
xmlns: = "URI"
	Stiamo specificando un namespace e il prefisso con cui ci riferiamo ad esso.
	Va definito come attributo in un elemento, e tutti i figli avranno quel namespace → idealmente va nella root del documento.
	Se non ha alcun prefisso allora stiamo specificando il **namespace di default. Tutti i figli saranno automaticamente associati ad esso.**
	
	`<root xmlns = "URI1", xmlns:prefisso ? "URI2"/>`
	`  <prefisso:elemento></prefiso:elemento>` → **URI2**
	`  <elemento></elemento>` → URI1
	`</root>`

In alto, lì insieme ai namespace, bisogna agiungere "**elementFormDefault="qualified**"" o non funziona nulla!

Tag principali

xsd:schema	è il nodo radice.
xsd:element	è il nodo per rappresentare gli elementi. Può contenere un altro elemento, oppure una <xsd:sequence> di elementi. *Attributi* › **<ref>** → Non avendo alcun prefisso, si sta riferenda all'elemento Book, che si trova nel namespace di default. › **<minOccurs>, <maxOccurs>**: di default sono a 1. › **<type>** permette di specificare il tipo di elemento – complex, simple o built in.
xsd:attribute	Si trova dentro un element e ne definisce un attributo.
xsd:simpleType	Definisce un custom tipo composto da "elementi singoli" (es. stringhe con restrizioni)
Xsd:complexType	Definisce un custom type composto da elementi complessi, quindi che hanno all'interno altri element o attributes.

Built-in data types

Esiston già una serie di tipi built-in, ma a volte non sono sufficienti:Problema: noi definiamo quasi sempre elementi stringa, ma in molti casi è fortemente insoddisfacente, come per esempio le date o il numero ISBN. Per rappresentarle in modo corretto vogliamo modificare lo schema di BookStore affinché anziché essere stringhe siano un tipo più adatto.

string	"Hello World"		**dateTime**	CCYY-MM-DD hh:mm:ss
boolean	{true,false,0,1}		**time**	hh:mm:ss.sss
decimal	7.08		**date**	CCYY-MM-Dd
float	12.56E3, INF, -INF, NAN, 0, -0		**gYearMonth**	CCYY-MM
double	12.56E3, INF, -INF, NAN, 0, -0		**gYear**	CCYY
duration	P1Y2M3DT10H30M12.3S		**gMonthDay**	--MM-DD

Custom data types

Le dichiarazioni possono essere:

- **Globali**: sono figli di xsd:schema → referenziabili da tutti gli elementi
- **Locali**: innestate in altri tipi → invisibili al resto dello schema

	simpleContent	complexContent
simpleType Può contenere solo tipo base con restrizioni/estensioni + attributo	Non si usa	Non si usa
complexType Può contenere figli e attributi	Sta estendendo/restringendo un tipo base	Sta estendendo/restringendo un altro complexType

simpleType

Può solo contenere ciò che estende/restringe (es. una stringa!), nuove regole (es. "solo stringhe LIKE %aa%) ed eventualmente attributi. È molto utile, ad esmepio, per avere una stringa a cui viene associato un ID.

Hanno solo restrizioni. Credo.

Ciascun tipo ha delle restrizioni aggiungibili:

Stringhe

Posso restringere tramite espressioni regolari, anche messe in OR con il separatore "|". I vincoli vengono estratti tramite facets da inserire all'interno:

<xsd:length value = "">	Lunghezza
<xsd:pattern value = "reg_expr">	Pattern
<xsd:enumeration value = "circle">	Enumerazione

```
<xsd simpleType name = "telephoneNumber">
        <xsd:restriction base = "xsd:string">
                <xsd:length value = "10"/>
                <xsd:pattern value = "REG_EXPR"/>
        </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name = "shape">
        <xsd:restriction base = "xsd:string">
                <xsd:enumeration value = "circle"/>
                <xsd:enumeration value = "square"/>
                <xsd:enumeration value = "triangle"/>
        </xsd:restriction>
</xsd:simpleType>
```

Integer

totalDigits	Lunghezza
pattern	Pattern

Enumeration	Enumerazioni
maxinclusive/maxEsclusive **minInclusive/minExclusive**	Limiti massimo e minimo.

complexTypes

Deve contenere o il tag complexContent o il tag simpleContent:

- simpleContent → estende o restringe un simpleType

- complexContent → estende o restringe un altro complexType

Extension	*Restriction*
Estende il complexType padre con altri **attributi o elementi**. Utilissimo l'id!	Permette di ridefinire gli elementi figli. Bisogna rimetterci anche tutti i valori che si trovavano nel tipo originario!

```
<xsd:element name = "città">
        <xsd:complexType>
                <xsd:simpleContent>
                        <xsd:extension
                base =”xsd:string”>
                                <xsd:attribute
                    name=”ctd_id”
                    type = “xsd:ID”
                    use = “required”/>
                            </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
</xsd:element>
```

```
<xsd complexType name = “ZeroAuthorPublication”>
        <xsd:complexContent>
                <xsd:restriction
                base = “Publication”>
                        <xsd:element name = “Title”
                    type = “” />
                        <xsd:element name = “Date”
                type = “xsd:gYear” />
                  </xsd:restriction>
        </xsd: complexContent >
</xsd:simpleType>
```

sequence	Sequenza.

choice	Per rappresentare delle alternative esiste l'elemento choice che permette di scegliere in modo esclusivo uno dei figli. SCELTE RIPETUTE: mi basta mettere choice ma aggiungendo maxoccurs = numero.
	
	`<xsd:complexType>`
	`  <xsd:choice>`
	`  <xsd:element ref="via">`
	`  <xsd:element ref="piazza">`
	`  <xsd:element ref="viale">`
	`  </xsd:choice>`
	`</xsd:complexType>`

all	ESPRIMERE CHE NON C'è UN ORDINE FISSO : Anziché sequence metto all. Tutti gli elementi devono avere maxOccur = 1. * Devono avere maxOccurs a 1 e minOccurs a 0 o 1. * Se estende un altro tipo, allora il tipo parent deve avere un contenuto vuoto * Non si può usare insieme a sequence o choice: i contenuti devono essere solo elementi

Proibire derivazioni

È utile in quanto potrei voler pubblicare un XML schema, ma voler garantire che solo i miei formati siano quelli ammissibili.

Per proibire le derivazioni uso l'attributo final nel tag complexType:

final = “all”	Vieta qualunque modifica
final = “restriction”	Vieta le restriction
final = “extension	Vieta le extension

Sostituzione di elementi: substitutionGroup

Può essere utile poter esprimere lo stesso elemento con più di un nome. Per esprimere quest fatto in XML usiamo la substitutiongroup, che consente di dichiarare un elemento e poi altri elemento che possono sistituirvisi. Attenzione: per poterli utilizzare devono essere dichiarati come globali!

Sono tipici di elementi customizzabili in base alla lingua: definiamo tutto in inglese e poi mettiamo elementi sostituibili in quanto in altre lingue.

```
<xsd:element name = “subway” type = “xsd:string”/>
<xsd:element name = “T” substitutionGroup = “subway” type = “xsd:string”/>
```

Attribute

Un attributo può avere solo tipi basi o restrizioni di tipi base → built-in types o sympleTypes.

La dichiarazione degli attributi è sempre l’ultima cosa, dopo la dichiarazione degli elementi.

Il tag <xsd:attribute>, a sua volta, può avere degli attributi

<xsd:attribute type = “…”/>	Indica il tipo. È simpleType oppure built-in (tipo xsd:string).
<xsd:attribute use = “…”/>	L’attributo uso ha senso solo nella dichiarazione di elementi: ad esempio, “per ciascun libro è richiesto l’attributo categoria”. Non si usa negli attributi globali. Indica il tipo di uso. Può essere: • required • optional • prohibited
<xsd:attribute default/fixed = “…”/>	Se è presente questo, use deve essere optional.

Per definire un elemento vuoto:

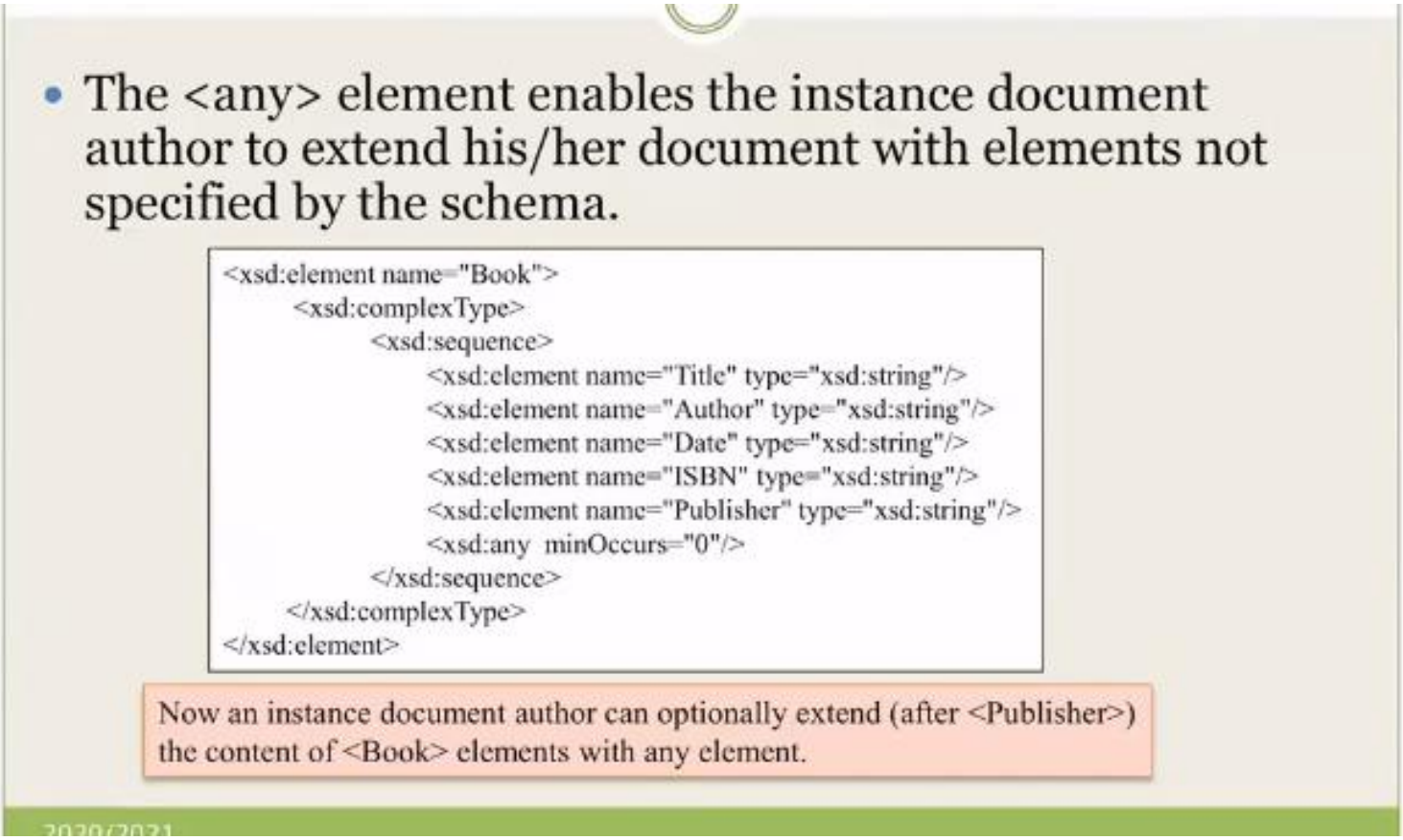
Schema:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="http://www.photography.org"
            xmlns="http://www.photography.org"
            elementFormDefault="qualified">
  <xsd:element name="gallery">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="image" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:attribute name="href" type="xsd:anyURI" use="required"/>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Instance doc (snippet):

```
<image href="http://www.xfront.com/InSubway.gif"/>
```

Elemento any



Commenti

<xsd:documentation>	Stringa di testo libera che dà commenti umani. Può avere i commenti: • **source** → nome/URL con la documentazione • **xml:lang** → lingua
<xsd:appinfo>	Commento per un programma; ad esempio può servire al validatore • **source** → nome/URL con la documentazione

Esercizio su XML

Rappresentare l'XSD relativo a questo XML.

1. **Specifico i namespace**.
   Me ne serviranno certamente almeno due: quello standard e quello relativo alle cose della banca.
   Per come lo abbiamo scritto, usiamo il prefisso xsd per lo schema e banca come namespace di default. Il target namespace può essere inventato :)
   ```
   <xsd:schema xmlns = "http://www.w3.org/2001/XMLSchema"
           targetNamespace = "www.banca.it"
           xmlns = "https://www.banca.it" >
   ```
2. **Definisco l'elemento banca** (radice del XML)
   Lo definisco come **tipo complesso** in quanto è una sequenza di elementi di tipo conto.
   Decidiamo di definire conto all'esterno e utilizzare il ref.
   ```
   <xsd:element name = "banca">
           <xsd:complexType>
                   <xsd:sequence>
                           <xsd:element ref="conto" maxOccurs = "unbounded"/>
                   </xsd:sequence>
           </complexType>
   </xsd:element>
   ```
3. **Definisco l'elemento conto**
   Anche qui abbiamo un insieme di sottoelementi, quindi useremo complextype.
   ```
   <xsd:element name = "conto">
           <xsd:complexType>
                   <xsd:sequence>
                           <xsd:element name = "num_conto" type = "xsd:unsignedInt"/>
                           <xsd:element name = "agenzia" type = "xsd:string"/>
                           <xsd:element name = "saldo" type = "xsd:unsignedInt"/>
                           <xsd:element ref = "cliente"/>
                   </xsd:sequence>
           </xsd:complexType>
   </xsd:element>
   ```

4. **Definisco il cliente, stavolte non come element ma solo come tipo (why???) e poi come elemento.**
   Non essendoci altri punti dove sfruttiamo questo tipo potevamo definirlo direttamente.
   ```
    <xsd:complexType name = "ClienteType">
                   <xsd:sequence>
                           <xsd:element name = "nome" type = "string"/>
                           <xsd:element name = "cognome" type = "xsd:string"/>
                           <xsd:element name = "via" type = "xsd:string"
                           <xsd:element name = "città" type = "xsd:string"/>
                   </xsd:sequence>
   </xsd:complexType>
   <xsd:element name = "cliente" type= "clienteType"/>
   </xsd:schema>
   ```

Limitare il problema di ridondanza.

Quello che si ripete nell'XML sono:

- I dati del cliente
- Ogni conto è fatto da un numero e da un'agenzia; l'agenzia è associata a più conti, quindi potrebbe essere ridondante.
- Città potrebbe essere condivisa

Quindi:

- Anziché rappresentre internamente tutti gli elementi, definiamo banca come sequenza di elementi ref (quindi, anche agenzia e città lo diventano)

```
<xsd:schema xmlns = "http://www.w3.org/2001/XMLSchema"
        targetNamespace = "www.banca.it"
        xmlns = "https://www.banca.it" />

<xsd:element name = "banca">
        <xsd:complexType>
                <xsd:sequence>
                        <xsd:element ref="città" maxOccurs = "unbounded"/>
                        <xsd:element ref="agenzia" maxOccurs = "unbounded"/>
                        <xsd:element ref="cliente" maxOccurs = "unbounded"/>
                        <xsd:element ref="conto" maxOccurs = "unbounded"/>
                </xsd:sequence>
        </xsd:complexType>
</xsd:element>
```

- Città sarà rappresentata come una stringa, ma per poterla riferire associamo anche un suo identificatore.

```
<xsd:element name = "città">
        <xsd:complexType>
                <xsd:simpleContent>
                        <xsd:extension base ="xsd:string">
                           <xsd:attribute name="ctd_id" type = "xsd:ID" use = "required"
                        </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
</xsd:element>
```

- Anche agenzia cambia

```
<xsd:element name = "agenzia">
        <xsd:complexType>
                <xsd:simpleContent>
                        <xsd:extension base ="xsd:string">
                           <xsd:attribute name="ag_id" type = "xsd:ID" use = "required"
                        </xsd:extension>
                </xsd:simpleContent>
        </xsd:complexType>
</xsd:element>
```

- Anche cliente cambia: sistemo il riferimento a città e anche per il cliente definiscoun ID che mi permetta di referenziare direttamente.

```
 <xsd:complexType name = "ClienteType">
                <xsd:sequence>
                        <xsd:element name = "nome" type = "string"/>
```

```
                    <xsd:element name = “cognome” type = “xsd:string”/>
                    <xsd:element name = “via” type = “xsd:string”
                    <xsd:element name = “città_cliente”>
                        <xsd:complexType>
                            <xsd:attribute name = “città_id”
                                                    type = “xsd:IDREF”
                                                    use = “required”/>
                        </xsd:complexType>
                    </xsd:element>
                </xsd:sequence>
                <xsd:attribute name = “cl_id” type = “xsd:ID” use = “required”/>
</xsd:complexType>

<xsd:element name = “cliente” type= “clienteType”/>
```

- Anche conto cambia

```
<xsd:element name = “conto”>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name = “numconto” type = “xsd:unsignedInt”/>
            <xsd:element name = “saldo” type = “xsd:unsignedInt”/>
            <xsd:element name = “agenzia_conto”>
                <xsd:attribute name = “agenzia_id” type = “xsd:IDREF”
                                                    use = “required” />
            </xsd:element>
        </xsd:sequence>
        <xsd:attribute name =intestatari” type = “xsd:IDREFS use = “required”/>
        <xsd:attribute name = “conto_id” tyoe = “xsd:ID” use = “required”/>
    </xsd:complexType>
</xsd:element>
</xsd:schema>
```