DATI SEMISTRUTTURATI

Vincenzo Calabrò

Dati semistrutturati

Basi di dati relazionali

- tutti i dati devono rispettare lo schema

Dati semistrutturati possono

- rispettare in modo parziale il proprio schema
- non avere schema predefinito

eXtended Markup Language

- formato di file proposto dal W3C per distribuire documenti elettronici (libri, manuali, cataloghi, …) sul World Wide Web

Concezione originale (1986)

- meta linguaggio per la specifica di linguaggi di markup

Oggi

- l'anello di congiunzione tra HTML e i sistemi informativi

XML, HTML e basi di dati

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, XML Schema)
- esistono linguaggi di query e trasformazione
- i dati sono neutri rispetto alle modalità di rappresentazione

XML vs HTML (1)

- HTML
  - insieme fisso di tag
- XML
  - standard per creare linguaggi di markup con tag personalizzati

Esempio

HTML

```
<h1>Basi di dati: compl</h1>
<ul>
 <li>P.Samarati
 <li>SSRI
 <li>online
</ul>
```

XML

```
<insegnamento>
 <titolo>
  Basi di dati: compl
 </titolo>
 <docente>P.Samarati</docente>
 <CdL>SSRI</CdL>
 <edizione>online</edizione>
</insegnamento>
```

XML vs HTML (2)

- XML non sostituisce HTML

- XML e HTML sono nati con scopi diversi
  - XML progettato per descrivere i dati ed evidenziare cosa rappresentano
  - HTML progettato per visualizzare i dati ed evidenziare come farli apparire

Uso di XML

- separare i dati da come vengono visualizzati
- scambiare i dati tra sistemi incompatibili
- scambiare informazioni in sistemi B2B
- condividere dati
- memorizzare dati
- supportare documenti nei moderni browser
  - è un documento di testo: il software che gestisce documenti testuali può gestire anche documenti XML

Sintassi XML (1)

- ogni documento XML deve avere un elemento tag radice
  - tutti gli altri elementi devono essere compresi nel tag radice
- tutti gli elementi hanno un tag di apertura e di chiusura
  - il tag di chiusura non può essere omesso (al contrario di HTML)
- i tag sono "case sensitive"
  - es., <a> diverso da <A> (al contrario di HTML)
- i tag devono essere annidati correttamente
  - es., <a>testo è sbagliato

Sintassi XML (2)

- ogni documento inizia con la dichiarazione di
  - versione di XML
  - tipo di codifica dei caratteri utilizzati nel file (opzionale)

Esempio

```
<?xml version="1.0" encoding="ISO-8859-1"?>
```

Elementi

- si possono estendere
- possono includere altri elementi (figli)
- hanno contenuto (che può essere vuoto)
- possono avere attributi

Esempio

```
<prodotto>
  <descrizione>Libro</descrizione>
  <prezzo>50</prezzo>
</prodotto>

<prodottovuoto></prodottovuoto>

<prodottovuoto/>
```

Attributi (1)

- contenuti negli elementi
- consentono di associare valore agli elementi senza essere considerati parte del loro contenuto
- i valori vanno racchiusi tra “ ”
- differiscono dagli elementi perché non possono contenere elementi figli

Esempio

```
<prodotto codice="123456">
  <descrizione>Libro</descrizione>
  <prezzo>50</prezzo>
</prodotto>
```

Attributi (2)

- hanno limitazioni
  - possono contenere un solo valore
    - non possono contenere valori multipli
  - non sono facilmente estendibili
  - non possono descrivere strutture
  - sono difficilmente manipolabili dalle applicazioni

- vanno bene per memorizzare metadati

Documento ben formato

Un documento è detto ben formato (well formed) se

- inizia con il prologo
  - es., `<?xml version="1.0"?>`
- tutti gli elementi hanno tag iniziali e finali
- la nidificazione dei tag è corretta
- gli attributi sono correttamente codificati
  - valori tra apici
- i valori sono correttamente codificati
  - case sensitive

Namespace (1)

- documenti diversi possono avere elementi/attributi con lo stesso nome

Esempio

```
<documento>
 <titolo>Lez 1</titolo>
 <testo>bla bla</testo>
</documento>
```

```
<persona>
 <titolo>dott</titolo>
 <nome>Rossi</nome>
</persona>
```

- usiamo prefisso per eliminare l'ambiguità

Esempio

```
<d:documento>
 <d:titolo>Lez 1</d:titolo>
 <d:testo>bla bla</d:testo>
</d:documento>
```

```
<p:persona>
 <p:titolo>dott</p:titolo>
 <p:nome>Rossi</p:nome>
</p:persona>
```

Namespace (2)

- collezione di nomi (attributi, elementi,…)
- identificato da un URI (Uniform Resource Identifier)
  - URI ha lo stesso formato di una URL ma non è una URL
  - rappresenta un nome non una locazione
- sintassi
  - `xmlns:prefisso="nomespazio"`
  - es., `xmlns:d="http://miospazioweb/d_schema"`

Namespace (3)

- un namespace associato ad un elemento si applica anche ai figli dell'elemento
- la dichiarazione del namespace nel prologo del documento si applica a tutti gli elementi del documento
- elementi fratelli possono avere namespace diversi

Esempio

```
<documento>
 <a:titolo>Lezione 1</a:titolo>
 <b:testo>bla bla</b:testo>
</documento>
```

DTD e XML Schema

Permettono di definire un modello per i documenti

- dettano il tipo dei documenti
  - i tag (elementi) ammessi e loro proprietà
    - cardinalità, attributi (dominio e valori di default), …
  - le regole di annidamento dei tag

Un documento che è validato rispetto a un DTD/XML schema è detto valido (valid)

Document Type Definition (DTD)

- elementi
  - definizione
  - cardinalità
- attributi

DTD: elementi

`<!ELEMENT … … >`

- contenenti altri elementi (figli)
  - es., `<!ELEMENT PRODOTTO (DESCRIZIONE)>`
- con PCDATA (parsed character data = testo)
  - es., `<!ELEMENT DESCRIZIONE (#PCDATA)>`
- contenuto misto
  - es., `<!ELEMENT ARTICOLO (#PCDATA|PRODOTTO)>`
- elementi vuoti
  - es., `<!ELEMENT ARTICOLO EMPTY>`

DTD: cardinalità di elementi

- 1 volta
  - es. `<!ELEMENT LISTA (PRODOTTO)>`
- 1 o più volte (+)
  - es., `<!ELEMENT LISTA (PRODOTTO+)>`
- 0 o più volte (*)
  - es., `<!ELEMENT LISTA (PRODOTTO*)>`
- 0 o 1 volta (?)
  - es.,`<!ELEMENT LISTA (PRODOTTO?)>`

DTD: attributi

<!ATTLIST … …>

- tipo (pochi, predefiniti, solo testuali)
  - ID, CDATA, IDREF, …
- vincoli
  - #REQUIRED: il valore deve essere specificato
  - #IMPLIED: il valore può mancare
  - #FIXED “valore”: se presente deve essere “valore”
  - “valore”: default se nessun valore è specificato

DTD: esempio

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

<elenco>
  <prodotto codice="123">
    <descrizione> libro </descrizione>
    <prezzo> 50 </prezzo>
  </prodotto>
  <prodotto codice="456">
    <descrizione> penna stilo </descrizione>
  </prodotto>
</elenco>
```

XML Schema (1)

Definisce gli elementi e la composizione di un documento XML

- inizialmente proposto da Microsoft
- divenuto W3C recommendation (maggio ‘01)
- noto anche come XSchema o XML Schema Definition (XSD)

XML Schema (2)

Vantaggi rispetto ai DTD

- sono estendibili
- sono file XML (i DTD non lo sono)
- sono più ricchi e completi
  - tipi di dati, elementi, ….
- gestiscono namespace

XML Schema (3)

- elementi semplici
- elementi complessi
- indicatori
  - ordinamento
  - cardinalità
  - raggruppamento
- attributi

XML Schema: elementi semplici (1)

```
<xs:element … … >
```

- non possono contenere altri elementi o attributi
- possono essere solo tipi standard (string, date, boolean, …) o tipi di dati derivati da questi
- possono avere valori di default (default)
- possono avere valori fissi (fixed)
- possono avere associate delle restrizioni (facet)
  - minimo e massimo, pattern di valori

XML Schema: elementi semplici (2)

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

Esempio

- definizione in XSD

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

- utilizzo in XML

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

XML Schema: restrizioni

RESTRIZIONE	DESCRIZIONE
enumeration	Lista di valori accettabili
fractionDigit	Numero di decimali dopo la virgola (>=0)
length	Numero esatto di caratteri (>=0)
min/maxExclusive	Limite numerico inferiore/superiore, esso escluso
min/maxInclusive	Limite numerico inferiore/superiore, esso compreso
min/maxLength	Lunghezza minima e massima
pattern	Sequenza di caratteri
totalDigit	Numero esatto di caratteri alfanumerici
whiteSpace	Tipo di gestione dei whitespace

Esempio

```
<xs:element name="sesso">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="maschio|femmina" />
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

XML Schema: elementi complessi

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

- possono contenere attributi o altri elementi
- possono utilizzare indicatori

XML Schema: indicatori

- ordinamento
  - `Any`: qualunque elemento, in qualunque ordine
  - `All`: tutti gli elementi, in qualunque ordine
  - `Choice`: uno e un solo elemento
  - `Sequence`: tutti gli elementi, nell'ordine specificato
- cardinalità (numero di occorrenze)
  - `maxOccurs`: max numero di occorrenze (default 1)
  - `minOccurs`: min numero di occorrenze (default 1)
- raggruppamento
  - `Group name`
  - `Group reference`

Elementi complessi: esempio

```
<xs:element name="persona">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="Cognome" type="xs:string" />
      <xs:element name="Nome" type="xs:string"
          maxOccurs="4" minOccurs="1" />
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XML Schema: attributi (1)

```
<xs:attribute … … >
```

- possono essere solo tipi standard (string, date, boolean, …)
- possono avere valori di default (default)
- possono avere valori fissi (fixed)
- possono essere obbligatori o opzionali
- possono avere associate delle restrizioni (facet)
  - minimo e massimo, pattern di valori

XML Schema: attributi (2)

```
<xs:attribute name="nome" type="tipo" />
<xs:attribute name="nome" type="tipo"
    default|fixed="xyz"
    use="required|optional" />
```

Esempio

- definizione in XSD

```
<xs:element name="cognome">
  <xs:complexType>
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="lang"
            type="xs:string" use="required" />
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:element>
```

- utilizzo in XML

```
<cognome lang="it"> Rossi </cognome>
```

XML Schema: esempio

```
<?xml version="1.0"?>
<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:attribute name="when" type="xs:date" use="required" />
</xs:element>
</schema>
```

```
<?xml version="1.0"?>
<note when="15/07/2006">
  <to> Rossi Andrea </to>
  <from> Bianchi Riccardo </from>
  <head> Promemoria </head>
  <body> Compleanno Michela </body>
</note>
```

XQuery

Permette di esprimere query su documenti XML

- proposta W3C: 15 Febbraio 2001
- basato su
  - XPath
  - espressioni FLWOR
    - clausole FOR, LET, WHERE, ORDER, RETURN

XPath

Permette di indirizzare parti di un documento

- documento XML modellato come albero di nodi
  - relazione contenimento figlio-padre
- una path expression seleziona oggetti che corrispondono ad un insieme di cammini sull'albero, ritorna
  - insieme di nodi (non ordinato, privo di duplicati)
  - valore booleano
  - numero (in virgola mobile)
  - stringa alfanumerica
- la valutazione di una path expression avviene in base al nodo contesto
  - `document(`*stringa_documento*`)` è la radice

XPath: sintassi

.	nodo corrente
..	nodo padre del nodo corrente
/	nodo radice, o figlio del nodo corrente
//	discendente del nodo corrente
@	attributo del nodo corrente
*	qualsiasi nodo
[ ]	predicato
[n]	posizione

Esempio

- seleziona l'attributo codice di tutti gli elementi libro figli di libreria

```
libreria/libro/@codice
```

Path expression: esempio (1)

Trova tutti i libri nell'elenco all'interno del documento `libri.xml`

```
document("libri.xml")/Elenco/Libro
```

```
<?xml version="1.0"?>
<Elenco>
 <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="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>
```

Path expression: esempio (2)

Trova tutti i titoli dei libri dell'editore Feltrinelli che sono disponibili e che si trovano nel documento `libri.xml`

```
document("libri.xml")/Elenco/Libro[Editore="Feltrinelli"
        AND @disponibilità="S"]/Titolo
```

```
<?xml version="1.0"?>
<Elenco>
 <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>
```

```
<Titolo>Il sospetto</Titolo>
```

Path expression: esempio (3)

Trova l'insieme di tutti gli autori che si trovano nel documento `libri.xml` annidati a qualunque livello

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

```
<?xml version="1.0"?>
<Elenco>
 <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>
```

```
<Autore>Umberto Eco</Autore>
<Autore>F. Dürrenmatt</Autore>
```

Path expression: esempio (4)

Trova gli elementi contenuti nel primo libro del documento `libri.xml`

```
document("libri.xml")/Elenco/Libro[1]/*
```

```
<?xml version="1.0"?>
<Elenco>
 <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>
```

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

Espressioni FLWOR (1)

- simili al costrutto `SELECT-FROM-WHERE` di SQL

- anziché essere applicate a tabelle associano variabili a valori e utilizzano tali associazioni per produrre i risultati

Espressioni FLWOR (2)

- `FOR`
  - per dichiarare variabili che permettono di iterare sugli elementi di un documento
- `LET`
  - per dichiarare variabili associate al risultato dell'espressione, eventualmente associandole a quelle introdotte con `for`
- `WHERE`
  - per esprimere condizioni (filtra le tuple prodotte dalla `for` o dalla `let`)
- `ORDER`
  - per ordinare le tuple (prodotte dalla `for` o dalla `let`)
- `RETURN`
  - per generare il risultato

For

- trova tutti i libri elencati nel documento libri.xml

```
FOR $lib IN document("libri.xml")//Libro
RETURN $lib
```

- trova tutti gli autori dei libri elencati nel documento libri.xml

```
FOR $lib IN document("libri.xml")//Libro
   FOR $auth IN $lib/Autore
RETURN $auth
```

Let

- trova tutti i libri elencati nel documento libri.xml

```
LET $lib := document("libri.xml")//Libro
RETURN $lib
```

Nota: produce un singolo binding per la variabile, l'intero insieme viene assegnato alla variabile

Where

- trova tutti i libri pubblicati da Bompiani che sono disponibili

```
FOR $lib IN document("libri.xml")//Libro
   WHERE $lib/Editore="Bompiani"
         AND $lib/@disponibilità="S"
RETURN $lib
```

Return (1)

- genera l'output di un'espressione FLWOR che può essere:
  - un nodo
    - `<Autore>F. Dürrenmatt</Autore>`
  - un foresta ordinata di nodi
    - `<Autore>Umberto Eco</Autore>`
      `<Autore>F. Dürrenmatt</Autore>`
  - un valore
    - `F. Dürrenmatt`
- può contenere
  - costruttori di elementi
  - riferimenti a variabili definite da `for` e `let`
  - espressioni annidate

Return (2)

- un costruttore di elemento ha
  - tag iniziale e tag finale
  - lista opzionale di espressioni che determinano il contenuto dell'elemento

Esempio

```
FOR $lib IN document("libri.xml")//Libro
WHERE $lib/Editore="Bompiani"
RETURN      <EdB>
                  $lib/Titolo
            </EdB>


<EdB><Titolo>Il nome della rosa</Titolo></EdB>
<EdB><Titolo>Il sospetto</Titolo></EdB>
```

Order by

Ordina il risultato della `return`

Esempio

Trova titolo ed editore dei libri elencati nel documento libri.xml, ritorna il risultato ordinato per titolo

```
FOR $lib IN document("libri.xml")//Libro
RETURN   <Libro>
                $lib/Titolo,
                $lib/Editore
         </Libro>
ORDER BY(Titolo ASCENDING)
```

XQuery: esempio (1)

```
<costruttore>
  <nome-co>Toyota</nome-co>
  <anno>2005</anno>
  <modello>
    <nome-mo>Prius</nome-mo>
    <cilindrata>1497</cilindrata>
    <potenza>77</potenza>
    <aliment>benzina</aliment>
  </modello>
  ...
</costruttore>
```

```
<macchina>
  <venditore>Mario Rossi</venditore>
  <marca>Toyota</marca>
  <anno>2005</anno>
  <modello>Prius</modello>
  <colore>grigio metalizz</colore>
  <prezzo>25000</prezzo>
  ...
</macchina>
```

XQuery: esempio (2)

```
document("costruttore.xml")
document("macchina.xml")
```

- realizzare un join a pari <nome-co>=<marca>, <nome-mo>=<modello> and <anno> = <anno>

```
FOR $c IN document("costruttore.xml")//costruttore,
    $m IN document("macchina.xml")//macchina,
    $mo IN $c/modello
  WHERE $c/anno=$m/anno
        AND $c/nome-co=$m/marca
        AND $mo/nome-mo=$m/modello
RETURN   <autovettura>
                $m/marca,
                $m/venditore,
                $m/modello,
                $mo/cilindrata,
                $m/prezzo
         </autovettura>
```

XQuery: esempio (3)

- trova gli editori che nell'elenco hanno almeno 100 libri

```
FOR $e IN document("libri.xml")//Libro/Editore
   LET $lib:=
       document("libri.xml")//Libro[Editore=$e]
   WHERE count($lib) > 100
RETURN $e
```

VINCENZO CALABRÒ

LinkedIn vincenzocalabro

www.vincenzocalabro.it