ROMA TRE
UNIVERSITÀ DEGLI STUDI

IBM DB2 “pureXML”

Complementi di basi di dati 2009

Andrea Gozzi – Fabrizio Celli

“pureXML”

IBM DB2 a partire dalla **versione 9.0** offre il supporto nativo alla memorizzazione di documenti XML all’interno di tabelle relazionali: tecnologia “pure XML”.

I **motivi** che hanno portato alla memorizzazione di documenti XML all’interno di database sono sostanzialmente gli stessi che hanno portato alla necessità di memorizzare nei database i dati relazionali:

- ricerca e recupero efficiente dei dati
- robusto supporto alla persistenza
- procedure backup e ripristino
- supporto per le transazioni
- performance e scalabilità

- Possibilità di integrazione tra documenti XML e dati relazionali già esistenti

- Possibilità di pubblicare dati relazionali in formato XML e viceversa

- Maggiore supporto alle applicazioni Web, SOA e Web Services, grazie all’interoperabilità offerta dallo standard XML

Vantaggi “pureXML”

1. Facile **riuso** del database preesistente: i documenti XML vengono registrati nelle colonne di tipo XML delle tabelle relazionali

2. Notevole aumento della **flessibilità**!

```
<DEPARTMENT deptid="15" deptname="Sales">
    <EMPLOYEE>
        <EMPNO>10</EMPNO>
        <FIRSTNAME>CHRISTINE</FIRSTNAME>
        <LASTNAME>SMITH</LASTNAME>
        <PHONE>408-463-4963</PHONE>
        <PHONE>415-010-1234</PHONE>
        <SALARY>52750.00</SALARY>
    </EMPLOYEE>
    <EMPLOYEE>
        <EMPNO>27</EMPNO>
        <FIRSTNAME>MICHAEL</FIRSTNAME>
        <LASTNAME>THOMPSON</LASTNAME>
        <PHONE>406-463-1234</PHONE>
        <SALARY>41250.00</SALARY>
    </EMPLOYEE>
</DEPARTMENT>
```

Requires:
- **Normalization of existing data !**
- **Modification of the mapping**
- **Change of applications**

Phone

EMPNO	PHONE
27	406-463-1234
10	415-010-1234
10	408-463-4963

Department

DEPTID	DEPTNAME
15	Sales

Costly!

Employee

DEPTID	EMPNO	FIRSTNAME	LASTNAME	PHONE	SALARY
15	27	MICHAEL	THOMPSON	406-463-1234	41250
15	10	CHRISTINE	SMITH	408-463-4963	52750

XML in DB2

IBM DB2 9.0 è il primo DBMS commerciale ad offrire il supporto nativo alla memorizzazione di documenti XML (“pure XML”). Consente di definire una colonna XML in una tabella non tipata.

```
CREATE TABLE clients(id int primary key not null, name
    varchar(50), status varchar(50), contact XML);
```

Ci sono due modi per inserire documenti XML in un database DB2:

1. Tramite dichiarazione **INSERT SQL**

```
INSERT into clients values(77,'John Smith','Gold',
   '<addr>111 Main St., Dallas, TX,0112</addr>')
```

2. Tramite utility **IMPORT**

```
IMPORT  from "C:\...\dir\clients.del" of del xml
        from "C:\...\dir" INSERT into clients
        (ID, NAME, STATUS, CONTACT);
```

“clients.del”:

3227, Ella Kimpton, Gold, <XSD FIL=’Client3227.xml’/>
8877, Chris Bontempo, Gold, <XSD FIL=’Client8877.xml’/>
…

XPath – Principi

XPath è un linguaggio utilizzato per **eseguire query su documenti XML** attraverso un database.

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

/dept/employee/phone

XPath - Espressioni

Due wildcard:

1. ***** → corrisponde a qualsiasi nome di elemento
2. ***//*** → indica un percorso arbitrario

Due simboli funzionali:

1. @ → specifica un attributo
2. **text()** → richiama soltanto il valore di un elemento

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

/dept/employee/@id

XPath - Espressioni

Due wildcard:

1. ***** → corrisponde a qualsiasi nome di elemento
2. ***//*** → indica un percorso arbitrario

Due simboli funzionali:

1. @ → specifica un attributo
2. **text()** → richiama soltanto il valore di un elemento

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

/dept/employee/name

XPath - Espressioni

Due wildcard:

1. ***** → corrisponde a qualsiasi nome di elemento
2. ***//*** → indica un percorso arbitrario

Due simboli funzionali:

1. @ → specifica un attributo
2. **text()** → richiama soltanto il valore di un elemento

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

/dept/employee/*/text()

XPath - Predicati

Con XPath è possibile specificare predicati, che possono essere visti come equivalenti alla clausola WHERE in SQL:

[@id = “902”] → WHERE id=‘902’

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

/dept/employee[@id=”902”]/name

XPath - Predicati

Con XPath è possibile specificare predicati, che possono essere visti come equivalenti alla clausola WHERE in SQL:

[@id = “902”] → WHERE id=‘902’

```
<dept bldg=“101”>
  <employee id=“901”>
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id=“902”>
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

//employee[office=”344” OR office=”216”]/@id

XPath - Predicati

Con XPath è possibile specificare predicati, che possono essere visti come equivalenti alla clausola WHERE in SQL:

[@id = “902”] → WHERE id=‘902’

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

/dept/employee[2]/@id

XPath – L'asse padre

Analogamente a MS-DOS e Linux/Unix si può usare un "**.**" nell'espressione XPath per indicare che si sta facendo riferimento al contesto attuale e un "**..**" per indicare un riferimento al contesto padre.

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

/dept/employee/name[../@id="902"]

XQuery

XQuery è un vero e proprio linguaggio di interrogazione indipendente dal linguaggio SQL, creato apposta per XML:

- supporta le **espressioni dei path** per navigare la struttura gerarchica di XML
- non supporta i **valori nulli** perché documenti XML omettono i dati mancanti o sconosciuti
- restituisce **sequenze di dati XML**
- supporta l'utilizzo della funzione **FLWOR** per la manipolazione dei documenti XML
- è **case sensitive** (come XPath)

XQuery – espressione FLWOR

FLWOR è una espressione che permette la manipolazione di documenti XML

- **FOR**: itera attraverso una sequenza, assegna una variabile agli oggetti
- **LET**: lega una variabile ad una sequenza
- **WHERE**: elimina gli elementi dell'iterazione
- **ORDER**: riordina gli elementi dell'iterazione
- **RETURN**: costruisce i risultati della query

XQuery – Esempi (1)

```
CREATE TABLE clients(id int primary key not null, name
    varchar(50), status varchar(50), contact XML);
```

“ Recupero dei documenti XML memorizzati in una colonna”

```
xquery db2-fn: xmlcolumn('CLIENTS.CONTACT')
```

```
<Client xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation="http://bogus">
  <Address>
    <street>
      5401 Julio Ave.
    </street>
    <city>
      San Jose
    </city>
…
```

XQuery – Esempi (2)

```
CREATE TABLE clients(id int primary key not null, name
    varchar(50), status varchar(50), contact XML);
```

“ Recupero dei dati XML relativi al fax dei clienti”

```
xquery
        for $y in db2-fn:xmlcolumn ('CLIENTS.CONTACT')/Client/fax
        return $y
```

```
<fax xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4087776666
</fax>
<fax xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
4085555555
</fax>
…
```

Si può sfruttare la potenza della clausola **return** per manipolare i risultati

XQuery – Esempi (3)

```
CREATE TABLE dept(deptID CHAR(8), deptdoc XML);
```

```
<dept bldg="101">
  <employee id="901">
    <name>John Doe</name>
    <phone>408 555 1212</phone>
    <office>344</office>
  </employee>
  <employee id="902">
    <name>Peter Pan</name>
    <phone>408 555 9918</phone>
    <office>216</office>
  </employee>
</dept>
```

```
xquery
for $d in db2-
fn:xmlcolumn('dept.deptdoc')/dept
let $emp := $d//employee/name
where $d/@bldg > 95
order by $d/@bldg
return
        <EmpList>
        {$d/@bldg, $emp}
        </EmpList>
```

```
<EmpList bldg="101">
      <name>John Doe </name>
      <name>Peter Pan </name>
</EmpList>
…..
```

DB2 SQL/XML

SQL/XML è il primo modo utilizzato per effettuare query su dati XML in DB2: il linguaggio principale rimane SQL, il quale viene arricchito con espressioni XQuery embedded.

Le **funzioni** offerte da SQL/XML sono disponibili con lo standard SQL 2006:

- *XMLPARSE*: effettua il parsing di caratteri o dati binari producendo dati XML
- *XMLSERIALIZE*: converte un valore XML in caratteri o dati binari
- *XMLVALIDATE*: valida il file XML rispetto uno schema definito
- *XMLEXISTS*: determina se una XQuery ritorna qualche risultato
- *XMLQUERY*: esegue una XQuery e ritorna la sequenza di risultati
- *XMLTABLE*: esegue una Xquery e ritorna i risultati come una tabella relazionale
- *XMLCAST*: fa il cast da o a un tipo di dato XML

SQL/XML – Funzioni (1)

```
CREATE TABLE clients(id int primary key not null, name
    varchar(50), status varchar(50), contact XML);
```

"Il nome dei clienti che vivono nell' area postale "95116""

Espressione XQuery

Inizializzazione di variabili

```
SELECT name
FROM clients
WHERE XMLEXISTS(
'$c/Client/Address[zip="95116"]' passing clients.contact as "c" )
```

Questa funzione testa se un'espressione XQuery ritorna una sequenza di uno o più oggetti: in tal caso viene restituito il valore booleano "true", altrimenti "false".
Si tenga inoltre presente che la clausola "`passing`" può contenere una sequenza di inizializzazioni di variabili (separate da virgola) e le variabili possono assumere anche valori contenuti in altre colonne della base di dati.

SQL/XML – Funzioni (2)

```
CREATE TABLE clients(id int primary key not null, name
    varchar(50), status varchar(50), contact XML);
```

"Elenco degli indirizzi di posta elettronica dei clienti con status "Gold""

```
SELECT XMLQUERY (
        '$c/Client/email' passing clients.contact as "c" )
FROM clients
WHERE status = 'Gold'
```

Questa funzione viene usata in genere nella clausola SELECT per selezionare una parte di un documento XML. E' simile a XMLEXISTS, solo che restituisce un valore XML (cioè un insieme di elementi, attributi e testo). Quindi l'esempio mostrato restituisce in output un documento XML.

SQL/XML – Funzioni (3)

```
CREATE TABLE items(id int primary key not null, brandname
varchar(30), itemname varchar(30), sku int, srp decimal(7,2),
comments XML);
```

“ Creazione di una tabella a partire da dati relazionali e dati XML”

```
SELECT t.comment#, i.itemname, t.customerID, t.message
FROM items i, XMLTABLE(
        '$c/Comments/Comment' passing i.comments as "c"
        columns
        comment# integer path 'CommentID',
        customerID integer path 'CustomerID',
        message varchar(100) path 'Message' ) as t
```

Questa funzione, usata in genere nella clausola FROM, genera una tabella a partire da un valore XML.

SQL/XML – Funzioni (4)

Le funzioni viste possono essere combinate per ottenere un'elevata espressività del linguaggio. Ad esempio, XMLQUERY consente di generare un documento XML a partire da uno esistente e inserirlo come valore di una colonna XML di una vista, mentre con XMLTABLE possiamo estrarre dallo stesso documento XML altri valori da inserire in colonne semplici della stessa vista.

```
CREATE VIEW P.PROVA (id, order)
AS
SELECT X.id,
   XMLQUERY('for $o in $c//shiporder
             return <shiporder>
                  {$o/@orderid, $o/shipto}
                  </shiporder>' passing S.order as "c")
FROM ORXSD.SHIPORDER as S,
   XMLTABLE ('$d//shiporder' passing S.order as "d"
   COLUMNS
   id integer path '@orderid') AS X;
```

Join con SQL/XML

```
CREATE TABLE dept (unitID CHAR(8), deptdoc XML);

CREATE TABLE unit(unitID CHAR(8) primary key not null, name
            CHAR(20), manager VARCHAR(20) … );
```

1° Metodo

```
SELECT u.unitID
FROM dept d, unit u
WHERE XMLEXISTS (
        '$e//employee[name=$m]'
        passing d.deptdoc as "e", u.manager as "m")
```

2° Metodo

```
SELECT u.unitID
FROM dept d, unit d
WHERE u.manager =
  XMLCAST (
    XMLQUERY (
      '$e//employee/name' passing d.deptdoc as "e") AS char(20) )
```

Join con XQuery

```
CREATE TABLE dept (unitID CHAR(8), deptdoc XML);

CREATE TABLE project (projectDoc XML);
```

```
xquery
   for $dept in db2-fn:xmlcolumn("DEPT.DEPTDOC")/dept
   for $proj in db2-fn:xmlcolumn("PROJECT.PROJECTDOC")/project
   where $dept/@deptID=$proj/@deptID
   return $dept/employee
```

Per eseguire un JOIN tra due colonne di tipo XML è preferibile utilizzare XQUERY, ma non necessario.

Introduzione a XML Schema

XML Schema Definition (**XSD**) è una alternativa XML-based al Document Type Definition (DTD) utilizzato per descrivere la struttura di un documento XML. In particolare definisce:

- Gli elementi e gli attributi che possono apparire in un documento
- Quali sono gli elementi figlio di un elemento
- Il numero e l'ordine degli elementi figlio di un elemento
- I tipi di dato per elementi e attributi
- I valori di default per elementi e attributi

...

Una delle caratteristiche più potenti di un XML Schema è il fatto che esso supporta i **tipi di dato**: con questo è più facile descrivere il contenuto ammissibile di un documento e validare la correttezza dei dati.

Un documento XML **ben formato** è un documento che è conforme alla sintassi XML. Anche se i documenti però sono ben formati,essi possono contenere errori, e questi errori possono avere conseguenze anche gravi: **un processo di validazione diviene piuttosto importante!**

XSD - Documento semplice

```
<?xml version="1.0"?>
<E>Hello</E>
```

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema>
        <xsd:element name="E" type="xsd:string"/>
</xsd:schema>
```

```
<?xml version="1.0"?>
<E xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:noNameSpaceSchemaLocation="schema.xsd">
Hello
</E>
```

Il tag **element** permette di definire un elemento del documento specificandone nome e tipo

Costrutti di base: Tipi semplici e complessi

In XML Schema ad ogni elemento si associa un tipo, che può essere semplice o complesso:

- Un **tipo semplice** è una stringa di testo non contenente altra marcatura
- Un **tipo complesso** può contenere al suo interno altri elementi e attributi

Un tipo complesso viene definito con il tag **`complexType`**

Costrutti di base: Tipi semplici

Gli elementi di tipo semplice sono quegli elementi che all'interno di un documento XML vengono rappresentati come **stringhe**.

E' possibile tuttavia restringere il campo delle stringhe valide. Per questo motivo sono stati definiti i **tipi semplici**, che sono:

- Numeri (es. float, double, decimal, int, byte, unsignedInt, positiveInt)
- Date (es. date, timeIstant, duration, gYear)
- Valori (es. ID, IDREF, ENTITY)
- Stringhe (es. string, normalizedString)
- Booleani (boolean)
- URI (uriReference)
- Dati binari

```
<xsd:complexType name="TipoCorso">
    <xsd:sequence>
        <xsd:element name="NOME" type="xsd:string"/>
        <xsd:element name="DOCENTE" type="TipoPersona"/>
        <xsd:element name="TUTOR" type="TipoPersona" .../>
        <xsd:element name="CDL" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>
```

Costrutti di base: Tipi complessi

Gli elementi complessi sono quelli associati a tipi che al loro interno definiscono **altri elementi e attributi**

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="CORSO" type="TipoCorso"/>
    <xsd:complexType name="TipoCorso">
        <xsd:sequence>
            <xsd:element name="NOME" type="xsd:string"/>
            <xsd:element name="DOCENTE" type="xsd:string"/>
            <xsd:element name="TUTOR" type="xsd:string"/>
            <xsd:element name="CDL" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:schema>
```

```
<CORSO>
  <NOME>Progetto di Linguaggi e Traduttori</NOME>
  <DOCENTE>Riccardo Rosati</DOCENTE>
  <TUTOR>Luigi Dragone</TUTOR>
  <CDL>Ing. Informatica</CDL>
<CORSO>
```

Costrutti di base: Cardinalità degli elementi

In XML Schema è possibile stabilire le cardinalità minime e massime dei vari elementi tramite l'uso degli attributi ***minOccurs*** e ***maxOccurs***, che possono assumere un qualsiasi valore numerico intero **>=0** oppure **"unbounded"** per indicare un numero illimitato (il default è 1 e 1 solo elemento)

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="CORSO" type="TipoCorso"/>
  <xsd:complexType name="TipoCorso">
    <xsd:sequence>
      <xsd:element name="NOME" type="xsd:string"/>
      <xsd:element name="DOCENTE" type="xsd:string"/>
      <xsd:element name="TUTOR" type="xsd:string" minOccurs="0"
      maxOccurs="unbounded"/>
      <xsd:element name="CDL" type="xsd:string"/>
      <xsd:element name="DURATA" type="xsd:string"/>
      <xsd:element name="ARGOMENTO" type="xsd:string" minOccurs="1"
      maxOccurs="unbounded"/>
      <xsd:element name="PROGETTO" type="xsd:string" minOccurs="0"
      maxOccurs="unbounded"/>
      <xsd:element name="CREDITI" type="xsd:string" minOccurs="0"
      maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Costrutti di base: Composizione di tipi complessi

Un tipo complesso può essere associato a un elemento durante la definizione di un ulteriore tipo complesso

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="CORSO" type="TipoCorso"/>
  <xsd:complexType name="TipoPersona">
    <xsd:sequence>
      <xsd:element name="NOMINATIVO" type="xsd:string"/>
      <xsd:element name="RECAPITO" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="TipoCorso">
    <xsd:sequence>
      <xsd:element name="NOME" type="xsd:string"/>
      <xsd:element name="DOCENTE" type="TipoPersona"/>
      <xsd:element name="TUTOR" type="TipoPersona" minOccurs="0"
      maxOccurs="unbounded"/>
      ...
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

Costrutti di base: Modalità di composizione di tipi complessi

Il raggruppamento degli elementi all'interno della definizione di un tipo complesso può avvenire in 3 modalità:

1. **Modalità sequenza** (elemento `sequence`): si indica il fatto che gli elementi nell'istanza devono apparire nello stesso ordine nel quale sono stati definiti nello schema (il numero di occorrenze di tali elementi deve essere in accordo con quanto specificato tramite gli attributi minOccurs e maxOccurs)

2. **Modalità arbitraria** (elemento `all`): si indica il fatto che gli elementi nell'istanza possono apparire in un ordine qualsiasi (il numero di occorrenze di tali elementi deve essere in accordo con quanto specificato tramite gli attributi minOccurs e maxOccurs)

3. **Modalità alternativa** (elemento `choice`): si indica il fatto che gli elementi nell'istanza devono apparire in alternativa, cioè o un elemento o un altro tra quelli definiti per il determinato tipo complesso (il numero di occorrenze dell'elemento candidato deve essere in accordo con quanto specificato tramite gli attributi minOccurs e maxOccurs, stavolta attributi dell'elemento `choice`)

Costrutti di base: Attributi

La definizione dei tipi complessi ammette la possibilità di specificare i relativi attributi.

Analogamente agli elementi anche gli attributi sono **tipati** ed è possibile specificare l'obbligatorietà o il valore di default:

```
<xsd:complexType name="TipoCorso">
  <xsd:sequence>
    <xsd:element name="NOME" type="xsd:string"/>
    ...
  </xsd:sequence>
  <xsd:attribute name="codice" type="xsd:string‟ use="required"/>
  <xsd:attribute name="homePage" type="xsd:uriReference"/>
</xsd:complexType>
```

Costrutti avanzati: Vincoli d'integrità

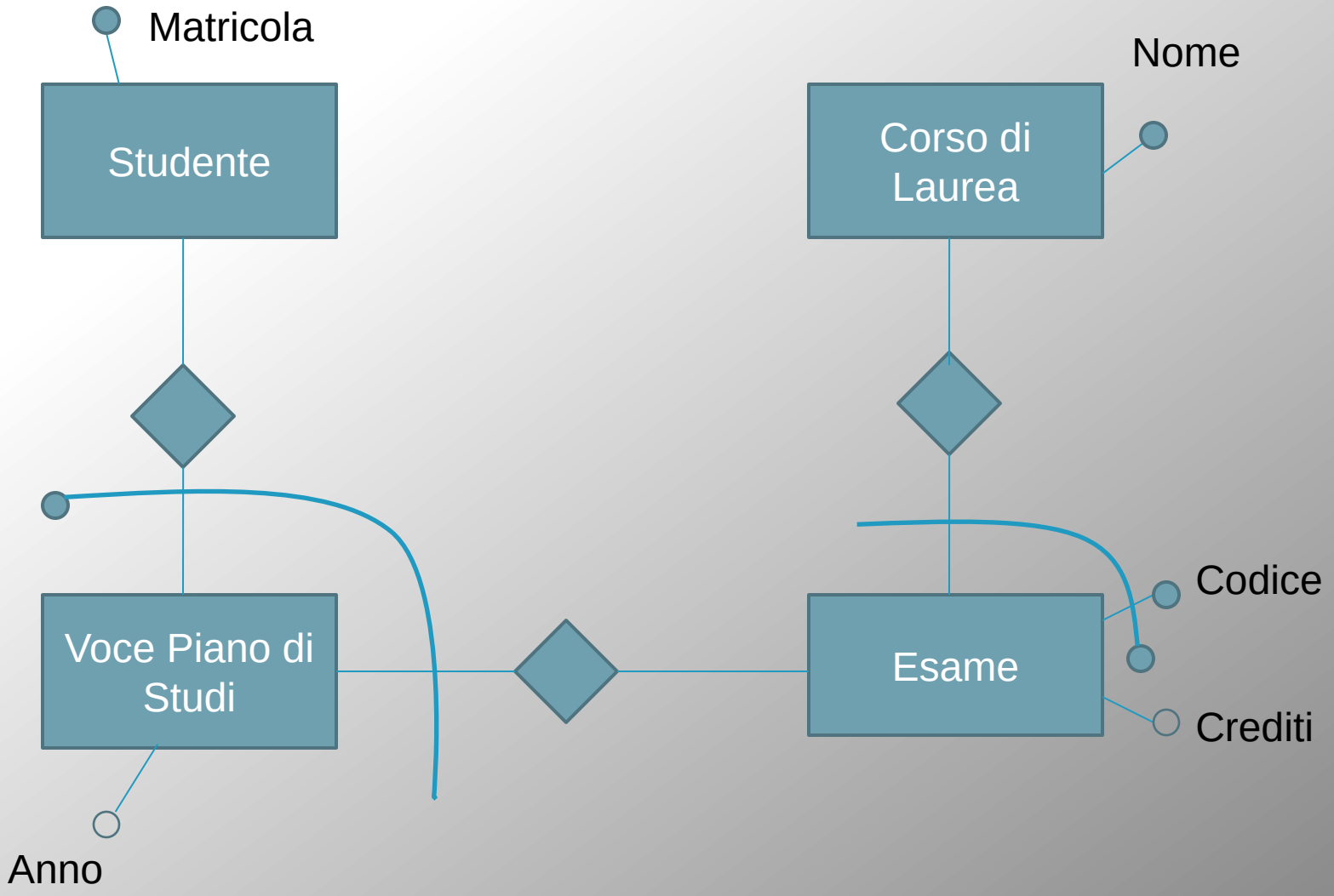
XML Schema introduce dei costrutti che permettono di definire vincoli abbastanza sofisticati anche per XML, basati sull'impiego di espressioni XPath:

- **Vincoli di unicità** (elemento **unique**)
- **Vincoli di chiave primaria** (elemento **key**)
- **Vincoli di chiave esterna** (elemento **keyRef**)

Tali vincoli sono validi nell'ambito di un singolo documento e sono caratterizzati da:

- Un **nome identificativo** (opzionale)
- Un **selettore** (elemento **selector**) che identifica mediante una espressione XPath gli elementi interessati
- Un **insieme di campi** (elemento **field**) che specifica i nodi interessati dal vincolo mediante espresssioni Xpath relative ai nodi identificati dal selettore
- Un **riferimento ad un altro vincolo** (nella definizione della chiave esterna bisogna indicare il nome della chiave primaria referenziata)

Costrutti avanzati: Esempio di vincoli d'integrità (1)



Costrutti avanzati: Esempio di vincoli d'integrità (2)

Studente(Matricola)
CdL(Nome)
Esame(NomeCDL, CodEsame, Crediti)
Pds(MatrStudente, NomeCdL, CodEsame, Anno)

```
<?xml version="1.0"?>
<xsd:schema:xsd=http://www.w3.org/2001/XMLSchema>

  <xsd:complexType name="TipoStudente">
    <xsd:sequence>
      <xsd:element name="PDS" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="anno" type="xsd:int"/>
          <xsd:attribute name="nomeCdL" type="xsd:string"/>
          <xsd:attribute name="codEsame" type="xsd:string"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="matricola" type="xsd:string"/>
  </xsd:complexType>
```

Costrutti avanzati: Esempio di vincoli d'integrità (3)

```
<xsd:complexType name="TipoCdL">
  <xsd:sequence>
    <xsd:element name="ESAME" minOccurs="0" maxOccurs="unbounded">
      <xsd:complexType>
        <xsd:attribute name="codice" type="xsd:string"/>
        <xsd:attribute name="crediti" type="xsd:int"/>
      </xsd:copmlexType>
    </xsd:element>
  </xsd:sequence>
  <xsd:attribute name="nome" type="xsd:string"/>
</xsd:complexType>
```

Costrutti avanzati: Esempio di vincoli d'integrità (4)

```
<xsd:element name="DBPDS">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="STUDENTE" type="TipoStudente"
      minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="CDL" type="TipoCDL" minOccurs="0"
      maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
```

Costrutti avanzati: Esempio di vincoli d'integrità (5)

Se ad esempio vogliamo definire un vincolo di **unicità** relativo agli attributi matricola, nomeCdL e codEsame dell'elemento STUDENTE, possiamo scrivere così:

```
<xsd:unique name="UniPDS">
   <xsd:selector xpath="./STUDENTE/PDS"/>
   <xsd:field xpath="../@matricola/">
   <xsd:filed xpath="@nomeCdL"/>
   <xsd:field xpath="@codEsame"/>
 </xsd:unique>
```

Costrutti avanzati: Esempio di vincoli d'integrità (6)

Se invece vogliamo esprimere un **vincolo di chiave esterna**, ad esempio quella tra il Piano di Studi e l'Esame:

1. Definire la **chiave primaria** relativa a ESAME. In realtà anche l'entità ESAME nello schema E-R aveva una chiave primaria ed una esterna, ma grazie al fatto che abbiamo definito un tipo complesso CDL che comprende ESAME si può definire la sola chiave primaria, comprensiva degli attributi di interesse:

```
<xsd:key name="PKESAME">
  <xsd:selector xpath="./CDL/ESAME"/>
  <xsd:field xpath="../@nome"/>
  <xsd:field xpath="@codice"/>
</xsd:key>
```

2. Definire il **vincolo di chiave esterna**, in riferimento alla chiave appena definita. Cioè poiché stavolta i due elementi sono distinti, c'è bisogno di un riferimento esterno:

```
<xsd:keyref name="FKPDS" refer="PKESAME">
  <xsd:selector xpath="./STUDENTE/PDS"/>
  <xsd:field xpath="@nomeCdL"/>
  <xsd:field xpath="@codEsame"/>
</xsd:keyref>
```

DB2 XML Schema Repository

IBM DB2 gestisce l' **XML Schema Repository (XSR)** , che è un repository per tutti i documenti che saranno necessari per validare e processare le istanze di documenti memorizzate in una colonna XML.

Le istanze di documenti XML contengono un riferimento a una **URI** che punta a un XML Schema associato: tale URI serve per validare l'istanza del documento.

DB2 gestisce le dipendenze tra le istanze di documenti XML e gli XML Schema relativi senza necessità di modifiche per quanto riguarda la URI.

Ogni database infatti contiene un XML Schema Repository che risiede nel catalogo.

Oggetti XSR

XSR supporta la creazione di una copia dell'informazione contenuta in un XML Schema

Tale informazione viene utilizzata per validare e processare le istanze di documenti XML memorizzate nelle colonne XML

Gli XML Schemas vengono identificati tramite degli **oggetti XSR**

Questi oggetti devono essere aggiunti al repository XSR tramite un **processo di registrazione** che può essere eseguito in diversi modi

Ogni XML Schema nel repository può consistere di **uno o più** documenti XML Schema

Registrazione di oggetti XSR

Il **processo di registrazione** di XML Schemas all'interno dell'XSR crea un oggetto XSR

1. Registrazione del documento XML Schema all'interno dell'XML Repository (XSR)
2. Specifica di addizionali schemi da includere nell'oggetto XSR (richiesta solo se l'XML Schema consiste di più di un documento)
3. Completamento del processo di registrazione

Può essere effettuato in **diversi modi**:

- Stored Procedures
- Command Line Processor
- Applicazioni Java

Registrazione tramite Stored Procedures

Le Stored Procedures necessarie alla registrazione di XML Schemas vengono create quando viene creato il database

1. Registrazione dell'XML Schema **primario**:

```
CALL SYSPROC.XSR_REGISTER ('user1', 'SOschema',
'http://mySOschema/SO', :content_host_var, NULL);
```

2. Aggiunta di uno **schema addizionale** da includere con lo schema primario:

```
CALL SYSPROC.XSR_ADDSCHEMADOC('user1', 'SOschema',
'http://mySOschema/address', :content_host_var, NULL);
```

3. **Completamento** della registrazione:

```
CALL SYSPROC.XSR_COMPLETE('user1', 'SOschema',
:schemaproperty_host_var, 0);
```

Registrazione tramite Command Line Processor

1. Registrazione dell' XML Schema **primario**:

```
REGISTER XMLSCHEMA 'http://mySOschema/SO'
FROM 'file://c:/TEMP/SO.xsd'
AS user1.SOschema
```

2. Aggiunta di uno **schema addizionale** da includere con lo schema primario:

```
ADD XMLSCHEMA DOCUMENT TO user1.SOschema
   ADD 'http://mySOschema/address'
   FROM 'file://c:/TEMP/address.xsd'
```

3. **Completamento** della registrazione:

```
COMPLETE XMLSCHEMA user1.SOschema
WITH 'file://c:TEMP/schemaProp.xml'
```

Supporto Java alla registrazione di XML Schemas

Il driver IBM DB2 per JDBC fornisce i metodi che consentono di eseguire le stesse funzioni tramite un programma Java

Per poter eseguire tali metodi le Stored Procedures devono essere state create all'interno del database (cosa che in DB2 avviene automaticamente al momento della creazione del database)

Elenco e informazioni relative agli XML Schemas registrati

Supponiamo di aver registrato tramite Command Line un XML Schema nel seguente modo:

```
REGISTER XMLSCHEMA ‘http://mySOschema/SO’
FROM ‘file://c:/TEMP/SO.xsd’
AS user1.SOschema

COMPLETE XMLSCHEMA user1.SOschema
```

Per recuperare tutti gli XML Schemas registrati con successo:

```
SELECT OBJECTNAME, OBJECTSCHEMA
FROM SYSCAT.XSROBJECTS
WHERE OBJECTTYPE='S' AND STATUS='C'
```

OBJECTNAME	OBJECTSCHEMA
SOSCHEMA	USER1

Recupero degli XML Schemas relativi a un documento XML

```
CREATE TABLE SHIPORDER(name varchar(20), order XML);
```

```
REGISTER XMLSCHEMA 'http://mySOschema/SO'
FROM 'file://c:/TEMP/SO.xsd'
AS user1.SOschema

COMPLETE XMLSCHEMA user1.SOschema
```

Per recuperare i documenti XML e gli XML Schemas associati:

```
SELECT ORDER, XMLXSROBJECTID(ORDER)
FROM SHIPORDER
```

ORDER	ID
XML …	281474976710656
XML …	281474976710656

Per identificare gli XML Schemas a partire dall'ID:

```
SELECT CAT.OBJECTSCHEMA, CAT.OBJECTNAME
FROM SYSCAT.XSROBJECTS AS CAT
WHERE CAT.OBJECTID = 281474976710656
```

OBJECTSCHEMA	OBJECTNAME
USER1	SOSCHEMA

Validazione di documenti XML rispetto a XML Schemas registrati

Una volta che un XML Schema è stato registrato è possibile riferirsi ad esso per validare i documenti XML che devono essere inseriti in apposite colonne XML

```
CREATE TABLE SHIPORDER(name varchar(20), order XML);

REGISTER XMLSCHEMA 'http://mySOschema/SO'
FROM 'file://c:/TEMP/SO.xsd'
AS user1.SOschema

COMPLETE XMLSCHEMA user1.SOschema

INSERT INTO SHIPORDER(name, order) VALUES ('Andrea',XMLVALIDATE
(XMLPARSE (DOCUMENT
'<?xml version="1.0" encoding="UTF-8"?>
<shiporder orderid="889923>
…
</shiporder>'
PRESERVE WHITESPACE )
ACCORDING TO XMLSCHEMA ID user1.SOschema ))
```

Validazione alternativa

E' possibile validare un documento XML tramite la "insert xmlvalidate" senza specificare il nome dello schema XSD nella query SQL tramite l'uso della seguente dichiarazione, da inserire nel file XML come attributo dell'elemento root:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation
="http://mySOschema/SO"
```

Evoluzione di un XML schema

Uno schema XSD che è stato registrato nell'XSR può essere modificato in un nuovo XSD compatibile col precedente senza validare nuovamente i documenti XML già inseriti nella base di dati. E' possibile usare a questo scopo la stored procedure XSR_UPDATE oppure il comando UPDATE XMLSCHEMA, alla condizione che il nuovo schema sia compatibile col precedente, altrimenti verrà sollevato un errore.
Dopo l'evoluzione, solo il nuovo XSD sarà disponibile, mentre le istanze XML e le loro URI rimarranno inalterate.

Vediamo alcuni criteri di compatibilità di schemi XML:

1. Gli <u>attributi</u> dichiarati o referenziati in un tipo complesso devono essere presenti anche nel nuovo XSD. Inoltre, gli attributi "required" possono essere presenti nel nuovo XSD solo se sono presenti nell'XSD originale.
2. La stessa cosa vale per gli <u>elementi</u>.

Evoluzione di un XML schema

3. Il tipo di un elemento o di un attributo nel nuovo XSD non è compatibile se i documenti XML già inseriti falliscono la validazione sul nuovo schema (ad esempio, se un elemento “a” era di tipo “string” e diventa “integer”)
4. Se il contenuto di un tipo complesso è dichiarato “mixed” nello schema originale, tale deve restare nel nuovo XSD
5. Se un attributo era definito “nillable”, deve restare tale nel nuovo XSD
6. Gli elementi globali dichiarati nell’XSD originale devono essere presenti anche nel nuovo XSD e non possono essere resi astratti
7. I tipi globali derivati da un altro tipo (ad esempio con un’estensione) devono essere presenti nel nuovo XSD
8. Un tipo complesso con contenuto semplice non può essere ridefinito con contenuto complesso
9. Un tipo semplice deve condividere lo stesso tipo di base

Evoluzione di un XML schema

Supponiamo di aver registrato il seguente XSD:

```
REGISTER XMLSCHEMA 'http://product'
FROM 'file://c:/schemas/prod.xsd'
AS STORE.PROD
COMPLETE XMLSCHEMA STORE.PROD;
```

Prima di poterlo aggiornare, il nuovo XSD deve essere registrato nell'XSR:

```
REGISTER XMLSCHEMA 'http://newproduct'
FROM 'file://c:/schemas/newprod.xsd'
AS STORE.NEWPROD
COMPLETE XMLSCHEMA STORE.NEWPROD

CALL SYSPROC.XSR_UPDATE(
     'STORE‘,'PROD','STORE','NEWPROD',1)
```