Bases de datos semi-estructurados

Dr. José Luis Zechinelli Martini
joseluis.zechinelli@udlap.mx
LIS – 3071

Administración de datos y de conocimiento
LAFMIA – UDLAP

Introducción

- Generaciones de bases de datos:
  - Red y jerárquico 70 - 80
  - Relacional 80 - 90
  - Objeto-relacional 90 - …
- WEB y bases de datos:
  - Pérdida de RDV (conexiones)
  - Servidores de aplicaciones débilmente acopladas
  - La WEB es una base distribuida muy voluminosa
  - Estructuración débil (flexible)
  - Orientado a documentos ...

XML

- Integración de datos y meta-datos
- Las bases de datos no pueden ser indiferentes:
  - Almacenamiento de documentos XML
  - Consulta de documentos XML
  - ¿Evolución o revolución?
- ¿Qué modelo de datos?
- ¿Qué lenguaje de consulta?
- ¿Cómo integrar soluciones nuevas y viejas?

Modelo interno y productos

- *Middleware* XML BD:
  - Arriba de un SGBD
  - Técnicas de mapeo sofisticadas
- Sistemas nativos:
  - Técnicas de investigación y almacenamiento especializado
- Extensión de SGBD relacionales:
  - Agregar tipos de datos nativos
  - Soportar documentos extendidos
- Bases de datos orientadas a objetos:
  - Uso de modelos orientados a objetos

Contenido

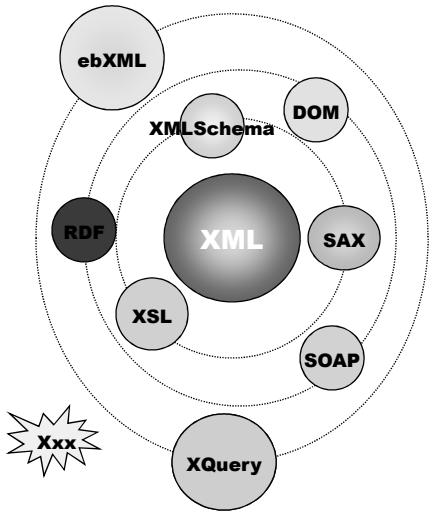
- Modelos de datos semi-estructurados:
  - ✓ Principio
  - – Introducción a XML
- Bases de datos XML:
  - – Conceptos de base
  - – DTD y esquemas XML
  - – XPath y XQuery
- Conclusiones

Introducción a XML

- XML es un meta-lenguaje universal para datos en la Web
- Permite el intercambio de contenido entre aplicaciones y/o entre navegadores
- XML apoya la estandarización de la manera de procesar la información:
  - – Intercambio (XML)
  - – Presentación (XSL)
  - – Recuperación (XQuery)
  - – Seguridad (Encriptación, Autentificación)
  - – Ligado (XLink)
  - – ...

La galaxia de estándares

- **XMLSchema**: esquema de documentos
- **XSL**: hojas de estilo
- **SAX**: API para la programación orientada a eventos
- **DOM**: API para la programación orientada a objetos
- **SOAP**: Protocolo de servicios Web
- **RDF**: Descripción de recursos Web
- **ebXML**: Estándares e-Commerce
- **Xxx**: Estándares de negocio



XML: Objetivos

- XML = un lenguaje de intercambio nuevo basado en *tags*
- XML = más simple que SGML
- XML = más complejo y eficiente que HTML
- XML = desarrollado por *XML Working Group* encabezado por W3C (desde 1996)
- XML 1.0 = recomendación oficial de W3C desde 10/02/1998

Elementos XML

- Elemento: Componente básico en XML
  - Delimitado por *tags*
  - Conteniendo texto (contexto de un elemento), otros elementos
- Ejemplo de elemento:

```
<person>
      <name> Alan </name>
      <age> 42 </age>
      <email> alan@abc.com </email>
</person>
```


Elementos XML

```
<table>
   <description> People on the fourth floor </description>
   <people>
       <person>
             <name> Alan </name>
             <age> 42 </age> <email> alan@abc.com </email>
       </person>
       <person>
             <name> Ryan </name>
             <age> 36 </age> <email> ryan@abc.com </email>
       </person>
       <person>
             <name> Patsy </name>
             <age> 58 </age> <email> patsy@abc.com </email>
       </person>
   </people>
</table>
```

HTML

```
<h1> People on the fourth floor </h1>

<p> <b> Alan </b>, 42 years,
    <i> alan@abc.com </i> </p>

<p> <b> Ryan </b>, 58 years,
    <i> ryan@abc.com </i> </p>

<p> <b> Patsy </b>, 36 years,
    <i> patsy@abc.com </i> </p>
```


Elementos XML

- Contenido del documento “People on the fourth floor”:
  - Nombres separados de las edades y de las direcciones *email*
  - Fácil de entender por cualquier aplicación
  - No hay información de cómo debe ser presentada
  - Texto del documento → PCDATA (*Parser Character Data*)
- Elementos vacíos:
  - `<married> </married>`
  - `<married/>`

Atributos XML

- Atributo:
  - Propiedad definida por la relación (nombre, valor)
  - Usado para especificar la lengua, el tipo de moneda, el formato, etc.
- Ejemplo:

```
<product>
    <name language="Spanish"> flauta transversal </name>
    <price currency="Pesos"> 4200.12 </price>
</product>
```


Atributos XML vs. Elementos XML

- Número de ocurrencias:
  - Un atributo puede ocurrir una sola vez
  - Un elemento puede repetirse varias veces
- Contenido:
  - El valor de un atributo es siempre una cadena
  - Un elemento puede contener sub-elementos

Problemas de ambigüedad

```
<person>
  <name> Alan </name>
  <age> 42 </age>
  <email> alan@abc.com </email>
</person>
```

```
<person
  name = "Alan"
  age = "42"
  email = "alan@abc.com"
/>
```

```
<person age = "42">
  <name> Alan </name>
  <email> alan@abc.com </email>
</person>
```


Documentos bien formados

- Documento bien formado:
  - *Tags* anidados apropiadamente
  - Atributos únicos
- El orden de los sub-elementos es relevante

Espacios de nombres

- ¿Cómo combinar *tags* provenientes de diferentes espacios?

Taxonomy = tag

```
<t xmlns:Guide="http://www.michelin.com/2001/Guide",
xmlns:Annuaire="http://www.pageblanche.com/2001/Guide">
```

- Mecanismos interesantes para integrar contenido

```
<Guide:Nom>Le Moulin</Guide:Nom>
<Annuaire:Nom>Le Moulin de Mougin</Annuaire:Nom>
```


Espacio de nombres

- Especificar globalmente nombres únicos para la definición de elementos
- Anteponer a cada etiqueta o atributo un identificador de recursos universal
- Usar un URL como identificador único:
  - Se puede definir una abreviatura para los identificadores
  - Se puede usar el atributo `xmlns` en el elemento raíz para definir un espacio de nombres predeterminado

Espacio de nombres: Ejemplo1

```
<table xmlns:SD = "http://www.SellsDepartment.hp.org">
   ...
   <SD:people>
       <SD:person>
             <SD:name> Alan </SD:name>
             <SD:age> 42 </SD:age>
             <SD:email> alan@abc.com </SD:email>
       </SD:person>
       ...
   </SD:people>
   ...
</table>
```


Espacio de nombres: Ejemplo2

```
<table>
   ...
   <people xmlns = "http://www.SellsDepartment.hp.org">
       <person>
             <name> Alan </name>
             <age> 42 </age>
             <email> alan@abc.com </email>
       </person>
       ...
   </people>
   ...
</table>
```

Contenido

- Modelos de datos semi-estructurados:
  - ✓ Principio
  - ✓ Introducción a XML
- Bases de datos XML:
  - Conceptos de base
  - DTD y esquemas XML
  - XPath y XQuery
- Conclusiones

Modelo XML basado en grafos

- Expresión XML:

```
<person>
    <name> Alan </name>
    <age> 42 </age>
    <email> alan@abc.com </email>
</person>
```

- Expresión SSD (*Semi-Structured Data*):

```
{person: {name: "Alan", age: 42, email: "alan@abc.com"}}
```

Referencias

- XML permite asociar un identificador a los elementos como valor de un atributo específico:

```
<state id = "s2">
    <scode> PUE </scode>
    <sname> Puebla </sname>
</state>

<city id = "c2">
    <cname> Ciudad de Puebla </cname>
    <state-of idref = "s2" />
</city>
```


Orden

```
<person>
    <firstname> John </firstname>
    <lastname> Smith </lastname>
</person>

<person>
    <lastname> Smith </lastname>
    <firstname> John </firstname>
</person>

<person firstname = "John" lastname = "Smith" />
<person lastname = "Smith" firstname = "John" />
```

Otros constructores

- Comentarios en XML:

```
<!-- this is a comment -->
```

- Instrucción de procesamiento:

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

<?xml-stylesheet      href = "book.css"
                      type = "text/css" ?>
```


Otros constructores

- CDATA: caracteres que podrían confundirse con marcas:

```
<![CDATA[<start> an incorrect element </end>]]>
```

- Macros tales como el carácter "<":

```
&lt;
```

- DTD:

```
<!DOCTYPE name [markupdeclarations]>
```

Otros constructores

- Documento completo en XML:

```
<?xml . . . ?>
<!DOCTYPE name [markupdeclarations]>
<name> . . . </name>
```

- Ejemplo:

```
<?xml version = "1.0" ?>
<!DOCTYPE db SYSTEM "person.dtd">
<db> <person> . . . </person> </db>
```


Contenido

- Modelos de datos semi-estructurados:
  - ✓ Principio
  - ✓ Introducción a XML
- Bases de datos XML:
  - ✓ Conceptos de base
  - – DTD y esquemas XML
  - – XPath y XQuery
- Conclusiones

DTD (*Document Type Definitions*)

- Servir de gramática para documentos XML
- Algunas extensiones sirven como esquemas para representar información
- Expresiones regulares:
  - **e*** (cualquier número de elementos)
  - **e+** (una o más ocurrencias)
  - **e?** (cero o una)
  - **e | e'** (alternancia)
  - **e , e'** (concatenación)

DTD: Gramáticas

```
<!DOCTYPE db [
      <!ELEMENT db (person*)>
      <!ELEMENT person (name,age,email)>
      <!ELEMENT name (#PCDATA)>
      <!ELEMENT age (#PCDATA)>
      <!ELEMENT email (#PCDATA)>
]>

<db>  <person>  <name> Alan </name>
                <age> 42 </age>
                <email> alan@abc.com </email>
      </person>
      . . .
</db>
```

DTD: Gramáticas

```
<!DOCTYPE recursiva [
    <!ELEMENT node (leaf | (node,node))>
    <!ELEMENT leaf (#PCDATA)>
]>

<node>
    <node>
          <node> <leaf> 1 </leaf> </node>
          <node> <leaf> 2 </leaf> </node>
    </node>
    <node>
          <leaf> 3 </leaf>
    </node>
</node>
```


DTD: Esquemas

Esquema relacional:

- R1(A: D1, B: D2, C: D3)
- R2(C: D3, D: D4)

R1

A	B	C
a1	b1	c1
a2	b2	c2

R2

C	D
c2	d2
c3	d3
c4	d4

DTD: Esquemas

```
<!DOCTYPE db [
    <!ELEMENT db (r1*, r2*)>
    <!ELEMENT r1 (a,b,c)>
    <!ELEMENT r2 (c,d)>
    <!ELEMENT a (#PCDATA)>
    <!ELEMENT b (#PCDATA)>
    <!ELEMENT c (#PCDATA)>
    <!ELEMENT d (#PCDATA)>
]>

<db> <r1> <a> a1 </a> <b> b1 </b> <c> c1 </c> </r1>
     <r1> <a> a2 </a> <b> b2 </b> <c> c2 </c> </r1>
     <r2> <c> c2 </c> <d> d2 </d> </r2>
     <r2> <c> c3 </c> <d> d3 </d> </r2>
     <r2> <c> c4 </c> <d> d4 </d> </r2>
</db>
```


DTD: Esquemas

- Permitir que los elementos de R1 y R2 aparezcan mezclados:

```
<!ELEMENT db ((r1|r2)*)>
```

- Describir componentes opcionales o repetidos:

```
<!ELEMENT r1 (a,b?,c+)>
```

- Almacenar la definición del esquema fuera del documento:

```
<!DOCTYPE db SYSTEM "schema.dtd">
<!DOCTYPE db SYSTEM "http://.../schema.dtd">
```

DTD: Atributos

```
<product>
  <name language = "Spanish" department = "music">
                        flauta transversal </name>

  <price currency = "Pesos"> 4200.12 </price>
</product>
```

```
<!ATTLIST name  language    CDATA #REQUIRED
                department  CDATA #IMPLIED>

<!ATTLIST price currency    CDATA #IMPLIED>
```


DTD: Referencias

```
<family>
    <person id = "jane" mother = "mary" father = "john">
          <name> Jane Doe </name>
    </person>
    <person id = "john" children = "jane jack">
          <name> John Doe </name>
    </person>
    <person id = "mary" children = "jane jack">
          <name> Mary Smith </name>
    </person>
    <person id = "jack" mother = "mary" father = "john">
          <name> Jack Smith </name>
    </person>
</family>
```

DTD: Referencias

- Redundancia:
  - Representación anidada de la información
  - Evitar reuniones de elementos para obtener la información asociada
- Producto cartesiano:
  - Representación normalizada
  - Combinar la información de la base de datos usando los identificadores

Documentos XML válidos

- Documento válido:
  - Bien formado
  - Conforme a una DTD
- Los identificadores deben ser valores distintos
- Los valores de las referencias deben ser identificadores existentes

Limitaciones de las DTD

- Imponen orden (usar "|" como alternativa)
- No hay noción de tipos atómicos: sólo el tipo #PCDATA
- No hay restricciones sobre las referencias: no se puede definir una referencia específica (ID, IDREF, IDREFS)
- Los *tags* son globales: usar espacios de nombres (`xmlns`), *v.g.*, `person:name` y `course:name`

Esquema XML (*XMLSchema*)

- Lenguaje de especificación de esquemas más sofisticado: resuelve muchas de las deficiencias de las DTD
- Cuenta con tipos de datos para restringir los elementos: `xsd:string`, `xsd:decimal`
- Permite indicar el número mínimo y máximo de apariciones de los sub-elementos:
  - Usando `minOccurs` y `maxOccurs`
  - Por defecto, `minOccurs = "1", maxOccurs = "1"`

DTD: Ejemplo

```
<!DOCTYPE db [
        <!ELEMENT db (person*)>
        <!ELEMENT person (name,age,email)>
        <!ELEMENT name (#PCDATA)>
        <!ELEMENT age (#PCDATA)>
        <!ELEMENT email (#PCDATA)>
]>

<db>    <person> <name> Alan </name>
           <age> 42 </age>
           <email> alan@abc.com </email>
        </person>
        . . .
</db>
```


Esquema XML: Ejemplo

```
<xsd:schema xmlns:xsd = "http://www.w3.org/2001/XMLSchema" >
<xsd:element name = "db" type = "Empleados"/>
<xsd:element name = "person">
   <xsd:element name = "name" type = "xsd:string"/>
   <xsd:element name = "age" type = "xsd:decimal"/>
   <xsd:element name = "email" type = "xsd:string"/>
</xsd:element>
<xsd:complexType name = "Empleados">
   <xsd:sequence>
         <xsd:element    ref = "person"
                         minOccurs = "0" maxOccurs = "unbounded"/>
   </xsd:sequence>
</xsd:complexType>
</xsd:schema>
```

Esquemas XML: Ventajas

- Permite crear tipos definidos por el usuario
- Permite restringir los textos de los elementos a tipos específicos: numérico, lista, etc.
- Permite restringir los tipos para crear tipos especializados: valor mínimo y máximo
- Permite la extensión de tipos complejos mediante el uso de una forma de herencia
- Permite restricciones de unicidad y de clave externa
- Está integrado con espacio de nombres para permitir a diferentes partes de un documento adaptarse a un esquema diferente

Contenido

- Modelos de datos semi-estructurados:
  - ✓ Principio
  - ✓ Introducción a XML
- Bases de datos XML:
  - ✓ Conceptos de base
  - ✓ DTD y esquemas XML
  - – XPath y XQuery
- Conclusiones

Consulta y transformación

- El resultado de una consulta XML puede ser un documento XML:
  - Extraer información de grandes volúmenes de datos
  - Convertir los datos entre distintas representaciones (esquemas) en XML
- Lenguajes de consulta y transformación:
  - XPath, expresiones de rutas de acceso (constructor)
  - XSLT, lenguaje de transformación (formato de datos)
  - XQuery, estándar para consultar datos XML

Modelo de árbol

- Nodos:
  - Atributos
  - Elementos que pueden tener nodos hijos
- Contenido textual de un nodo:

```
<element>
   Éste es un <bold> buen </bold> libro
</element>
```

- Orden de los elementos y atributos

XPath

- Expresiones de ruta de acceso:
  - Secuencia de pasos de ubicación separados por “/” (en lugar del “.” de OQL)
  - El resultado es un conjunto de valores
- Ejemplo: `/family/person/name`

/family/person/name

```
<family>
    <person id = “jane” mother = “mary” father = “john”>
          <name> Jane Doe </name>
    </person>
    <person id = “john” children = “jane jack”>
          <name> John Doe </name>
    </person>
    <person id = “mary” children = “jane jack”>
          <name> Mary Smith </name>
    </person>
    <person id = “jack” mother = “mary” father = “john”>
          <name> Jack Smith </name>
    </person>
</family>
```

/family/person/name

```
<family>
    <person id = "jane" mother = "mary" father = "john">
          <name> Jane Smith </name>
    </person>
    <person id = "john" children = "jane jack">
          <name> John Smith </name>
    </person>
    <person id = "mary" children = "jane jack">
          <name> Mary Smith </name>
    </person>
    <person id = "jack" mother = "mary" father = "john">
          <name> Jack Smith </name>
    </person>
</family>
```


XPath

- Expresiones de ruta de acceso:
  - Secuencia de pasos de ubicación separados por "/" (en lugar del "." de OQL)
  - El resultado es un conjunto de valores
- Ejemplo: `/family/person/name`

```
<name> Jane Doe </name>
<name> John Doe </name>
<name> Mary Smith </name>
<name> Jack Smith </name>
```

XPath

- Acceder a los valores de los elementos:

```
/family/person/name/text()
```

- Acceder a los valores de los atributos:

```
/family/person/@children
```

- De forma predeterminada, no se siguen las referencias IDREF

XPath

- Expresión regular:

  Nombre de todas las personas

```
/family//name
```

- Acceder a través de un índice a un atributo:

  Hijos de la primera persona

```
/family/person[1]/@children
```

XPath

- Predicados de selección:
  `/family/person[ age > 18 ]`
- Contar nodos coincidentes:
  `/family[ count(person) > 2 ]`
- Saltar niveles (`//`)
- La función "`id`"
- El operador "`|`" (unión)
- Padre (`/..`) y descendientes (`//`)

XPath

Selector	Nodos seleccionados
/	Raíz del documento
//	Saltar niveles o descendientes
*	Cualquier elemento
nombre	Elemento de *tag* "nombre"
@*	Todos los atributos
@nombre	Atributo de nombre "nombre"
text()	Cualquier nodo texto
processing-instruction('nombre')	Instrucción de procesamiento "nombre"
comment()	Cualquier nodo comentario
node()	Cualquier nodo
id('valor')	Elemento cuyo id es "valor"

XQuery

- Consultas:
  - Parecidas a las consultas SQL
  - Organizadas en expresiones FLWR
- Cuatro secciones:
  - **for**: similar a la cláusula **from** de SQL
  - **let**: asignación de expresiones a variables
  - **where**: similar a la cláusula **where** de SQL
  - **return**: construcción de resultados en XML

XQuery: Consulta simple

- Obtener las referencias de los hijos de los adultos mayores:

```
for $p in /family/person
let $lista := $p/@children
where $p/age >= 60
return <hijos> {$lista} </hijos>
```

```
for $p in /family/person[ age > 60 ]
return <hijos> {$p/@children} </hijos>
```

XQuery: Ordenar

- Usar al final de cualquier expresión la cláusula `order by`:

  - ```
    for $p in /family/person
    order by $p/name
    return <result> {$p/*} </result>
    ```

  - ```
    order by $p/name descending
    ```


XQuery: Funciones

- Funciones sobre conjuntos:
  - Eliminar duplicados: `distinct-value`
  - Funciones de agregación: `sum`, `count`, etc.
- El operador "`->`" se puede aplicar sobre valores de tipo:
  - IDREF para obtener el elemento
  - IDREFS para obtener un conjunto de elementos

XQuery: Funciones

Lista con los nombres de las personas con el promedio de edades de sus hijos:

```
{       FOR $p IN /family/person
        LET $e := avg(/family/person/@children->person/age)
        RETURN
        <resultado>
                {$p/name}
                <promedio> {$e} </promedio>
        </resultado>
}
```


XQuery: Funciones

- Funciones incorporadas:
  - `document(name)`
  - `number(string)`
- Otras características:
  - `if-then-else`
  - `some $e in path satisfies P`
  - `every $e in path satisfies P`

XQuery: Funciones

Dar un documento con **encabezado**, **título** y **lista** de personas con todos sus hijos mayores y otro con las personas con al menos un hijo menor de edad:

```
<documento>
    <personas-hijos-mayores>
        <título>Lista de las personas con todos sus hijos mayores de edad</título>
        { FOR $p IN /family/person
          WHERE EVERY e in $p/@children->person/age SATISFIES ( e > 18 )
          RETURN $p }
    </personas-hijos-mayores>
    <personas-con-hijos-menores>
        <título>Lista de las personas con hijos menores de edad</título>
        { FOR $p IN /family/person
          WHERE SOME e in $p/@children->person/age SATISFIES ( e < 18 )
          RETURN $p }
    </personas-con-hijos-menores>
    <fecha> {current-date()} </fecha>
</documento>
```


Contenido

- Modelos de datos semi-estructurados:
  - ✓ Principio
  - ✓ Introducción a XML
- Bases de datos XML:
  - ✓ Conceptos de base
  - ✓ DTD y esquemas XML
  - ✓ XPath y XQuery
- Conclusiones

Conclusiones

- ¿XML puede cambiar la construcción de bases de datos?
  - Investigación en BD semi-estructuradas
  - Necesidad de esquemas flexibles (XML Schema)
  - Lenguajes de consulta estandarizados (XQuery)
  - El efecto de la Web ...
- ¿Integración débil usando objeto-relacional?
  - Transformación a tablas
  - Administración de grafos
  - ¿Middleware o SGBD?

¿?