Teórico 9:
XML

XML

- Estructura de datos XML.
- Esquemas de documentos XML.
- Consultas y transformaciones.
- APIs para XML.
- Almacenamiento de datos XML.

Introducción

- XML: Extensible Markup Language.
- Definido por la WWW Consortium (W3C).
- Puede representar datos de base de datos como muchas clases de datos estructurados.
- Es muy utilizado para comunicar o integrar aplicaciones.
- Los documentos tienen marcas dando información adicional sobre las secciones del documento.
  - Ej. <titulo> XML </titulo> <transparencia> Introducción …</ transparencia >

XML Introducción (Cont.)

La capacidad de especificar nuevas etiquetas, y de crear las estructuras anidadas hace a XML una técnica muy apropiada para intercambio de datos.

- Las marcas hacen datos (relativamente) autodocumentados.
  - Ejemplo:

```
<supermercado>
    <articulo>
        <nro_articulo> 10 </ nro_articulo >
        <descripcion> Harina </descripcion >
        <precio> 20.8 </precio>
    </articulo>
    <proveedor>
        <nro_proveedor> 202  </nro_proveedor>
        <nombre> Molinos cañuelas </nombre>
    </proveedor>
</supermercado>
```

XML: Motivación

- Hoy en día el intercambio de datos entre aplicaciones es crítico.
  - Ejemplos:
    - Bancos: transferencias de fondos entre cuentas de distintos bancos.
    - Procesamiento de ordenes entre varias empresas.
    - Lenguaje muy utilizado en Web Services, el protocolo SOAP está basado en XML.

- Cada aplicación tiene su propia forma de representar la información.

- XML se ha convertido en la base para todos los formatos de intercambio de datos de nueva generación.

XML Motivación (Cont.)

- Fácil generación de formatos basados en texto plano con headers que indican el significado de los campos

- Cada tipo de documento basado en XML define cuáles son los elementos válidos, usando:
  - Tipos de lenguajes para especificar la sintaxis:
    - DTD (Document Type Descriptors).
    - XML Schema.
  - Se puede agregar una descripción textual de la semántica de una marca.

- XML permite definir nuevas marcas cuando se lo requiere, sin embargo esto puede ser restringido por el archivo de metadatos (DTDs o XML Schema).

- Una gran variedad de herramientas está disponible para analizar, mostrar y consultar documentos de XML/datos.

Comparación con Datos Relacionales

- Ineficiente: Las marcas se repiten en cada fila de una tabla

- Mejor que datos relacionales a la hora intercambio de datos.

Estructura de Datos XML

- **Marca (Tag)**: label para una sección de datos.
- **Elemento (Element)**: sección de datos que comienza con <*tagname*> y termina con </*tagname*>.
- Los elementos pueden ser anidados.
  - Anidación correcta
    - <articulo> … <precio> …. </precio> </articulo>
  - Anidación incorrecta
    - <articulo> … <precio> …. </articulo> </precio>
  - Formalmente: cada marca de comienzo (<marca>) deben tener una única marca de finalización </marca>.
- Todo documento XML debe tener un único elemento Raíz.

Ejemplo de Elementos Anidados

```
<concesionaria>
    <auto>
        <patente> ADB210 </patente>
        <modelo> 2005 </modelo >
        <propietario>
            <dni> 24564857 </dni>
            <apellido> Lopez </apellido>
            <nombre> Carlos </nombre>
        </propietario>
        <propietario>
          ……..
        </propietario>
    </auto>
…
  </concesionaria>
```

Estructura de Datos XML (Cont.)

- Mezclar texto con subelementos está permitido en XML.
  - Ejemplo:

```
<auto>
    Este auto no esta asegurado
        <patente> ADB210 </patente>
        <modelo> 2005 </modelo >
</auto>
```

  - Es útil para representación de documentos, pero no para representación de datos.

Atributos

- Los elementos pueden tener **atributos.**
- Los atributos se especifican con el par <nombre> = <*valor*> dentro de la marca de comienzo de un elemento.
- Un elemento puede tener varios atributos pero cada atributo sólo puede estar una sola vez en el elemento.
- **Ejemplo**

```
<auto estado="bueno" puertas="5">
        <patente> ADB210 </patente>
        <modelo> 2005 </modelo >
</auto>
```

Espacio de nombres

- Los datos XML tienen que ser intercambiados entre las organizaciones.
- El mismo nombre para una marca puede tener diferente significado en diferentes organizaciones, esto puede causar confusión a la hora de intercambiar documentos.
- Especificando un String único como nombre de elemento evita confusión.
- Usando XML Namespaces se evita tener que utilizar nombre largos para garantizar la unicidad

```
<supermercado Xmlns:SM='http://www.supermercado.com'>
    …
     <SM:articulo>
        <SM:nro_cuenta>10</SM:nro_cuenta> …
     </SM:articulo>

        …
</supermercado>
```

Otras Características Sintácticas

- Los elementos sin subelementos o contenido de texto pueden ser abreviados finalizando el elemento de datos con />, de esta forma no se debe especificar el final.
  - <auto estado="bueno" puerta="5" patente = "DFS356" modelo = "2004" />
- Para almacenar string de datos que contienen marcas, sin que estas sean interpretadas como subelemento, se debe utilizar CDATA de la siguiente manera:
  - <![CDATA[<articulo> … </articulo>]]>

  De esta forma <articulo> y </articulo> son interpretado como texto y no como marcas.

Esquema de Documento XML

- Los esquemas de bases de datos restringen la información que pueden almacenar en cuanto a cantidad y tipo de datos.
- Los documentos XML no requieren tener asociado un esquema que defina su estructura.
- Sin embargo, los esquemas son muy importantes para el intercambio de datos XML.
- Existen 2 mecanismos para especificar esquemas XML
  - **Document Type Definition (DTD)**
    - Muy utilizado antes que surgiera XML Schema.
  - **XML Schema**
    - Más reciente, más utilizado en la actualidad.

Document Type Definition (DTD)

- El tipo de un documento XML puede ser especificado utilizando DTD.
- Un DTD restringe la estructura de los datos XML o sea que elementos puede tener.
- Que atributos debe o puede tener un elemento.
- Que subelements debe o puede tener incluido cada elemento y cuantas veces.
- Un DTD no restringe el tipo de datos, todos los valores son string en XML.
- Sintaxis DTD
  - <!ELEMENT element (subelements-specification) >
  - <!ATTLIST element (attributes) >

Especificación Elementos en DTD

- Los subelementos pueden ser especificados de la siguiente manera:
  - Nombre de los subelementos, o
  - #PCDATA (un string),
  - EMPTY (no tiene subelementos) o ANY (cualquier subelemento)
- Ejemplo

```
<!ELEMENT cuenta ( número-cuenta nombre-sucursal saldo )>
<!ELEMENT cliente ( nombre-cliente calle-cliente ciudad-cliente )>
<!ELEMENT número-cuenta ( #PCDATA )>
.....
<!ELEMENT ciudad-cliente( #PCDATA )>
```

- La especificación de subelementos pueden tener expresiones regulares

```
<!ELEMENT banco ( (cuenta |cliente| impositor)+)>
```

  - Notación:
    - "|" - alternativas
    - "+" - 1 o mas ocurrencias
    - "*" - 0 o mas ocurrencias

DTD de un Banco

- <!DOCTYPE banco [

  <!ELEMENT banco ((cuenta |cliente| titular)+)>

  <!ELEMENT cuenta (número-cuenta nombre-sucursal saldo)>

  <!ELEMENT cliente (nombre-cliente calle-cliente ciudad-cliente)>

  <!ELEMENT titular (nombre-cliente número-cuenta)>

  <!ELEMENT número-cuenta (#PCDATA)>

  <!ELEMENT nombre-sucursal (#PCDATA)>

  <!ELEMENT saldo(#PCDATA)>

  <!ELEMENT nombre-cliente(#PCDATA)>

  <!ELEMENT calle-cliente(#PCDATA)>

  <!ELEMENT ciudad-cliente(#PCDATA)>

] >

Especificación de Atributos en DTD

- Para cada atributo se debe especificar:
  - Nombre
  - Tipos de atributos:
    - CDATA
    - ID (identificador)
    - IDREF (ID referencia)
    - IDREFS (multiples IDREFs)
  - Los valores de los atributos pueden ser:
    - #REQUIRED (obligatorios)
    - “valor”(valor por defecto),
    - #IMPLIED (no tiene valor predeterminado y es opcional)
- Ejemplos de especificación de atributos
  - <!ATTLIST cuenta tipo-cuenta CDATA “corriente” > //tiene valor por defecto “corriente”
  - <!ATTLIST cliente

    cliente-id ID #REQUIRED

    cuentas IDREFS #REQUIRED >

DTD de Banco con Atributos

- DTD de Banco con identificadores y referencias.

```
<!DOCTYPE banco-2 [
    <!ELEMENT cuenta ( sucursal, saldo )>
    <!ATTLIST cuenta
      número-cuenta ID #REQUIRED
      titulares IDREFS #REQUIRED >
    <!ELEMENT cliente ( nombre-cliente, calle-cliente, ciudad-
      cliente )>
    <!ATTLIST cliente
      cliente-id ID #REQUIRED
      cuentas IDREFS #REQUIRED >
    ⋯ declaraciones para sucursal, saldo, nombre-cliente,
    calle-cliente y ciudad-cliente · · ·
] >
```

Datos XML con Identificadores y Referencias

```
<banco-2>
    <cuenta número-cuenta = “C-401” titulares = “C100 C102”>
        <nombre-sucursal> Centro </nombre-sucursal>
        <saldo> 500 </saldo>
    </cuenta>
    ...
    <cliente cliente-id = “C101” cuentas = “C-402”»>
        <nombre-cliente>Jorge</nombre-cliente>
        <calle-cliente> Montaña </calle-cliente>
        <ciudad-cliente> Córdoba </ciudad-cliente>
    </cliente>
    <cliente cliente-id = “C102” cuentas = “C-401 C-402”>
        <nombre-cliente>María</nombre-cliente>
        <calle-cliente> Bs As</calle-cliente>
        <ciudad-cliente> Río Cuarto </ciudad-cliente>
     </cliente>
</banco-2>
```

Limitaciones de los DTDs

- No existen los tipos, los valores son todos String.
- IDs y IDREFs no tienen tipos.

XML Schema

- XML Schema es un lenguaje más sofisticado que está siendo el reemplazo de los DTDs. XML Schema soporta:
  - Tipos de datos.
  - Restricciones de valores mínimos y máximos.
  - Tipos de datos definidos por el usuario.
  - Ademas soporta claves, claves foráneas, herencia, etc.
- XML Schema se especifican en sintaxis XML,
- XML Schema está integrada con espacio de nombres.
- XML Schema más complejo que los DTDs.

Versión XML Schema del DTD de Banco

```
<xs:schema xmlns:xs=http://www.w3.org/2001/XMLSchema>
<xs:element name="banco" type="tipoBanco"/>
<xs:element name="cuenta">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="nro_cuenta"   type="xs:string"/>
            <xs:element name="sucursal"      type="xs:string"/>
            <xs:element name="saldo"         type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>
….. definición de cliente
<xs:complexType name="tipoBanco">
    <xs:sequence>
        <xs:element ref="cuenta" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="cliente" minOccurs="0" maxOccurs="unbounded"/>
        <xs:element ref="titular" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
</xs:complexType>
</xs:schema>
```

Otras características XML Schema

- Los atributos se especifican con la marca xs:attribute:
  - <xs:attribute name = “numero_cuenta”/>
  - Con use = “required” se especifica la obligatoriedad del valor.
- Restricciones de clave: Ejemplo la definición de nro_cuenta como clave de cuenta del esquema banco:

```
<xs:key name = “claveCuenta”>
        <xs:selector xpath = “/banco/cuenta”/>
        <xs:field xpath = “nro_cuenta”/>
<\xs:key>
```

- Restricción de clave foránea: Ejemplo el titular de una cuenta:

```
<xs:keyref name = “fkCuenta” refer=“claveCuenta”>
        <xs:selector xpath = “/banco/titular”/>
        <xs:field xpath = “numero_cuenta”/>
<\xs:keyref>
```

Consultas y Transformaciones de Datos XML

- Transformación de información de un esquema XML a otro.
- Consultas en datos en formato XML.
- Lenguajes estándares de consulta y transformación:
  - XPath
    - Lenguaje que consiste en expresiones de tipo “path”.
  - XQuery
    - Un lenguaje de query para datos XML que tiene muchas características interesante.
  - XSLT
    - Lenguaje para transformaciones de XML a XML y de XML a HTML.

Modelo de Árbol de los Datos XML

- Los lenguajes de consulta y transformaciones están basados en el modelo árbol de los datos XML.
- Un documento XML está modelado como un árbol, donde los nodos son elementos, atributos y textos:
  - Los nodos elementos pueden tener hijos, los cuales pueden ser subelementos o atributos.
  - El texto de un elemento se modela como un nodo hijo del elemento.
  - Los hijos de un nodo se ordenan de acuerdo al orden en el documento XML.

XPath

- XPath se utiliza para seleccionar de un documento utilizando **“expresiones path”.**
- Una expresión path es una secuencia de pasos separados por “/”, similar a la ruta (path) de un archivo en un sistema de archivos.
- El resultado de una expresion path es el conjunto de nodos que machean con el path de la expresion.
- Por ejemplo. /banco-2/cliente/nombre-cliente evaluado en el *banco-2* visto en una transparencia anterior es:

  <nombre-cliente>Jorge</nombre-cliente>

  <nombre-cliente>Maria</nombre-cliente>
- Otro ejemplo: /banco-2/cliente/nombre-cliente/text( )

  Retorna el mismo resultado que el ejemplo anterior pero sin los tag.

- Su especificación en:

  http://www.w3.org/TR/2007/REC-xpath20-20070123/

XPath (Cont.)

- "/" hace referencia a la raíz del documento.
- Las expresiones Path se evalúan de izquierda a derecha, cada paso opera sobre el conjunto de valores del paso previo
- Los predicados de selección pueden estar en cualquier paso de la expresión, se especifican con [ ]
  - Ejemplo 1: /banco-2/cuenta[saldo]
    - Retorna los elementos cuenta que contienen un subelemento saldo.
  - Ejemplo 2: /banco-2/cuenta[saldo > 400]
    - Retorna los elementos cuentas con saldo >400.
- Los atributos se acceden con "@"
  - Ejemplo: /banco/cuenta[saldo > 400]/@numero-cuenta
    - Retorna los numero de cuenta de las cuentas con saldo >400.

Funciones XPath

- XPath provee varias funciones
  - La función count() cuenta la cantidad de elementos generados por el path
    - Ejemplo /banco-2/cuenta[count(./clientes) > 2]
      - Retorna las cuentas con más de 2 clientes

- Los operadores lógicos and , or y la función not() se puede utilizar en los predicados.

- IDREFs se pueden referenciar utilizando la función id()
  - id() Se puede aplicar a un conjunto de referencias, como ser IDREFS y en un string que contienen múltiples referencias separadas por blancos.
  - Ejemplo: /banco-2/cuenta/id(@titulares)
    - Retorna todos los cliente referenciados desde el atributo titulares del elemento cuenta.

Más características XPath

- Se utiliza el operador "|" para realizar la unión del resultado de dos expresiones, pero "|" no puede ser anidado con otros operadores.
- Se utiliza "//" para saltar varios niveles de nodos
  - Ejemplo: /banco-2//nombre-cliente
    - Encuentra el atributo nombre de cliente de cualquier elemento que lo posea bajo el XML banco-2.
- La función doc(<nombre>) retorna la raíz de un documento.

- Especificación completa de XPATH en:

  http://www.w3.org/TR/2007/REC-xpath20-20070123/

XQuery

- XQuery es un lenguaje de propósito general para datos XML.
- Está estandarizado por la World Wide Web Consortium (W3C)

  http://www.w3.org/TR/2007/REC-xquery-20070123/

- Xquery fue derivado de Quilt query language, Xquery tienen caracteristicas de SQL, XQL and XML-QL
- XQuery utiliza expresiones de la siguiente forma

  for … let … where … order by …result …

- Equivalencia con SQL

  for ⇔ SQL **from**
  where ⇔ SQL **where**
  order by ⇔ SQL **order by**

  return ⇔ SQL **select**
  let permite variables temporales, esto no tiene equivalencia en in SQL

Sintaxis FLWOR en XQuery

- La sintaxis de estas expresiones se puede encontrar en la especificación XQuery en:

  http://www.w3.org/TR/2007/REC-xquery-20070123/#id-flwor-expressions

- En las cláusulas se utilizan expresiones XPath y variables para almacenar los resultados de las expresiones Xpath.
- Ejemplo de expresión FLWOR en XQuery:
  - En encontrar los números de cuenta de todas las cuentas con saldo > 400, el resultado debe contener cada número de cuenta encerrada con la marca <numero-cuenta> .. </ numero-cuenta >

```
for     $x in /banco-2/cuenta
let     $nroCuenta := $x/@numero-cuenta
where $x/saldo > 400
return <numero-cuenta> {$nroCuenta } </ numero-cuenta >
```

  - En la cláusula **return** el texto encerrado entre { } son evaluados.
- La expresión del ejemplo anterior se podría escribir sin la cláusula let de la siguiente manera:

```
for $x in /banco-2/cuenta[saldo>400]
return <numero-cuenta> { $x/@numero-cuenta } </numero-cuenta >
```

Joins

- Los Joins se especifican de manera similar a SQL

```
for $a  in  /banco/cuenta,
    $c  in  /banco/cliente,
    $d  in  /banco/titular

where   $a/numero-cuenta = $d/numero-cuenta
    and $c/nombre-cliente = $d/nombre-cliente
return <cliente-cuenta> { $c $a } </cliente-cuenta>
```

- La misma consulta puede ser expresada solamente con las cláusulas for y return :

```
 for $a  in  /banco/cuenta,
     $c  in  /banco/cliente,
     $d  in  /banco/titular [
       $a/numero-cuenta = $d/numero-cuenta
       and $c/nombre-cliente = $d/nombre-cliente
                              ]
   return <cliente-cuenta> { $c $a } </cliente-cuenta>
```

Queries Anidados

- El siguiente query transforma los datos con estructura plana de la figura 10.1 de [SILV 06] en los datos de la estructura anidada de las figura 10.4 de [SILV 06]

```
<banco-1> {
    for $c in /banco/cliente
    return
       <cliente>
         { $c/* }
         { for $d in /banco/impositor[nombre-cliente = $c/nombre-
        cliente],
                $a in /banco/cuenta[numero-cuenta = $d/numero-
        cuenta]
          return $a }
       </cliente>
} </banco-1>
```

- $c/* denota todos los hijos de $c.

Sorting in XQuery

- La cláusula **order by** puede ser utilizada al final de cualquier expresión, por ejemplo:

```
for $c in /banco/cliente
order by $c/nombre-cliente
        return <cliente> { $c/* } </cliente>
```

- Por defecto el orden es descendente.

Funciones y otras características de XQuery

- Los usuarios pueden crear con el sistema de tipos de XMLSchema declarar una función:
- ```
  function saldo($c as xs:string) as xs:decimal* {
    for $d in /banco/titular[nombre-cliente = $c],
        $a in /banco/cuenta[numero-cuenta = $d/numero-cuenta]
    return $a/saldo
  }
  ```
- Los tipos son opcionales para los parámetros y retorno de las funciones.
- El * luego del tipo indica una secuencia de valores del tipo.
- XQuery provee los cuantificadores universales y existenciales en los predicados de la cláusula where.
  - **some** $e **in** *path* **satisfies** *P*
  - **every** $e **in** *path* **satisfies** *P*
- XQuery soporta cláusula If-then-else.
  ```

Transformaciones XSLT

- Una hoja de estilo almacena opciones de formato para documentos, que generalmente esta separado del documento. Por ejemplo en una hoja de estilo HTML se puede especificar el color y tamaño de la fuente de letra, etc.

- La **XML Stylesheet Language (XSL)** fue originalmente diseñado para generar código HTML a partir de XML

- XSLT es un lenguaje de transformación de propósito general, puede transformar XML a XML, y XML a HTML

- Las transformaciones XSLT son expresadas utilizando reglas llamadas **templates.** Los **templates** combinan selecciones usando XPath con construcción de resultados.

- Especificación disponible en http://www.w3.org/TR/xslt20/

Templates XSLT

- Ejemplo de un template XSLT que hace un “macheo” y un select

```
<xsl:template match=“/banco-2/cliente”>
        <xsl:value-of select=“nombre_cliente”/>
</xsl:template>
```

El atributo “match” de xsl:template especifica un patrón en sintaxis XPath.

- Los elementos en el documento XML que machean con el patron son procesadas por la acción definida en el elemento xsl:template.
  - xsl:value-of select (salidas) de los valores especificados (en el ejemplo, nombre-cliente)
- Para los elementos que no “machean” con ningún template
  - Sus atributos y texto es generado en la salida.

Creación de una salida XML

- Toda marca o texto en una hoja de estilo XSL que no es parte del espacio de nombre xsl se genera como salida sin ningún cambio.
- El siguiente ejemplo genera una salida “marcada” de todos los clientes del banco (cambiando la estructura) :

```
<xsl:template match=“/banco-2/cliente”>
    <cliente>
    <xsl:value-of select=“nombre_cliente”/>
    </cliente >
</xsl:template>
```

  - La salida del ejemplo

```
<cliente>Jorge</cliente>
<cliente>Maria</cliente>
```

Creando Salida XML (Cont.)

- Para crear atributos nuevos para un elemento se debe utilizar el constructor xsl:attribute
  - Ejemplo

```
<cliente>
  <xsl:attribute name="id_cliente">
    <xsl:value-of select = "id_cliente"/>
  </xsl:attribute>
</cliente>
```

La salida será

```
<cliente  id_cliente="…."> ….
```

APIs

- Hay dos APIs para datos XML :
  - **SAX** (Simple API para XML)
    - Basado en modelo de parser, provee manejadores de eventos y eventos de parser.
  - **DOM** (Document Object Model)
    - Los datos **XML** son parseados y transformados en un árbol.
    - Variedad de funciones para navegar el árbol DOM.
    - Ejemplo: Java DOM API provee la clase Node con los siguientes metodos:

      getParentNode( ), getFirstChild( ), getNextSibling( )
      getAttribute( ), getData( ) (for text node)
      getElementsByTagName( ), …
    - También, provee funciones para actualizar el árbol.

Almacenamiento de Datos XML

- Los datos XML pueden ser almacenados en:
  - Almacenamiento no relacionales
    - Archivos planos
    - Base de datos XML
      - Soportan el modelo DOM
      - Todavía no están maduros
  - Base de datos relacionales
    - Los datos se deben traducir al modelo relacional.
    - Ventaja, Los RDBMS están muy maduros.
    - Desventaja: overhead de traducción de datos y queries

Almacenamiento de Datos XML en Base de Datos Relacionales

- Hay tres alternativas:
  - Representación String.
  - Representación de árbol.
  - Mapeo a relaciones.

Representación String

- Los elemento de datos se almacenan como cadenas en una relación.

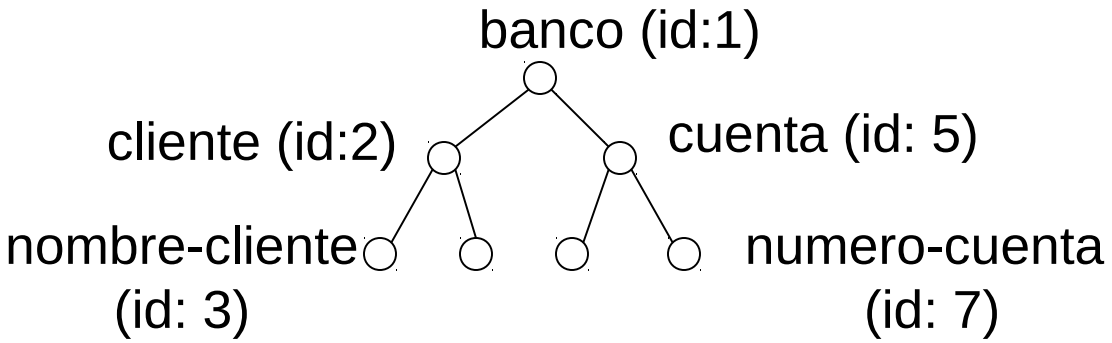
- No se almacena la estructura.

Representación de Árbol

- **Representación de Árbol:** El modelo de árbol XML se almacena utilizando dos relaciones

  nodos(id, tipo, rotulo, valor)

  hijos (id_hijo, id_padre)



Mapping de Datos XML a Relacional

- Se crea una relación por cada tipo de elemento cuyo esquema es el siguiente:
  - Un atributo id como identificador único del elemento
  - Un atributo en la relación para cada atributo del elemento
  - Un atributo id_padre para tener una referencia al padre del elemento
  - También se puede almacenar su posición (nro de hijo)
- Todos los subelementos que aparecen solo una vez pueden ser almacenado como atributos de la relación

Bibliografía Principal

- [SILV 10]Database System Concepts. Sixth Edition, Edition Silberschatz, Korth, Sudarshan. McGraw Hill Company, 2010.
- [SILV 06]Fundamentos de Base de Datos. Quinta Edición, Edition Silberschatz, Korth, Sudarshan. McGraw Hill Company, 2006.