Bases de Datos II

Francisco Javier Mercader Martínez

Índice

Tema 1: Recuperación de datos y formatos de serialización

1.1) Necesidad de formatos de serialización

- Los formatos de **serialización** son vitales para el intercambio de datos
- Más en el ámbito *Big Data* (atacan a la "V" de la *variabilidad*)
- A lo largo de los años se han diseñado formatos de serialización
- Algunos son **más eficientes** que otros.
- Algunos se han **estandarizado**.
- En cualquier caso, son fundamentales para transmitir información, ya sea a trave´s de **ficheros de disco** o bien para **comunicación por red**.
- La mayoría de los formatos son **de propósito general**, por lo que una misma información se pude codificar **de varias formas** (por ejemplo, como `INSERT` en SQL, ficheros CSV, etc.)
- Nuestro objetivo es conocer los formatos más usados para elegir el correcto en cada ocasión.

1.2) Características a analizar

- **¿Es un protocolo estándar?-** Con ello nos referimos a si está avalado por cuerpo de estándares o su especificación
-

- Tabla resumen

	Estándar	Bin?	Humano?	Ref?	IDL?	Ext?	API?
Apache Avro	Sí	Sí	No	N/A	Sí (acoplado)	Sí	N/A
CSV	Parcial (RFC4180)	No	Sí	No	No	Parcial	No
JSON	Sí (RFC7159)	No (BSON)	Sí	Sí (RFC6901)	Parcial	Sí	No
Thrift	No	Sí	Parcial	No	Sí (acoplado)		
XML							

1.2.1) Pandas

- La herramienta más conocida y usada de Pandas son los `DataFrames`.
- Permite almacenar datos tabulares en dos dimensiones similar a una hoja de cálculo o una base de datos relacional.
- Las columnas de datos

```
df = pd.DataFrame(rand)
```

1.2.2) XML (eXtensible Markup Language)

- Meta-lenguaje de etiquetas derivado de SGML.
- Motivación:
  - Intercambio de datos en Internet
- Reúne los requisitos de un lenguaje de intercambio de información:

  - Simple: al estar basado en etiquetas y legible
  - Independiente de la plataforma: codificación UNICODE
  - Estándar y amplia difusión: W3C
  - Definición de estructuras complejas: DTD, Schemas
  - Validación y transformación: DTD, XSLT
  - Integración con otros sistemas
- Facilita procesamiento lado cliente.
- **No muy utilizado para BigData. Ha perdido tracción, es demasiado complejo finalmente y es muy poco eficiente en cual al porcentaje datos/metadatos**

Ejemplo

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2  <DISCO CODIGO="B000067FSG">
3    <TITULO>Estrella de Mar</TITULO>
4    <ARTISTA>Amaral</ARTISTA>
5    <ESTILO>Pop</ESTILO>
6    <REFERENCIA>
7      <EDITORA>Virgin</EDITORA>
8      <AÑO_EDICION>2002<AÑO_EDICION>
9    </REFERENCIA>
10   <MUSICOS>
11     <MUSICO ROL="cantante">Amaral</MUSICO>
12     <MUSICO ROL="guitarra">Juan Aguirre</MUSICO>
13   </MUSICOS>
14 </DISCO>
```

- Instrucciones de procesamiento (línea 1)
- Raíz (línea 2)
- Etiquetas y atributos

- DTD

```
1  <!ELEMENT DISCO (TITULO, ARTISTA, ESTILO?, REFERENCIA, MUSICOS)>
2  <!ATTLIST DISCO CODIGO ID #REQUIERED>
3  <!ATTLIST DISCO TIPO=(CD | LP | DVD) "CD">
4  <!ELEMENT TITULO (#PCDATA)>
5  ...
6  <!ELEMENT REFERENCIA (EDITORA, AÑO_EDICION) >
7  <!ELEMENT MUSICOS (MUSICO*)>
8  ...
```

Describe los documentos XML $\Longrightarrow$ **Validación**

- DTD (ii)

Documento XML:

- **Válido:** sigue la estructura de un DTD

```
<!DOCTYPE web-app
  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
```

- **Bien formado:** Sigue las reglas de XML

Limitaciones:

- No es XML
- Tipado limitado de datos
- No soporta espacios de nombres

- API SAX

Librería **`xml.sax`**

```
import xml.sax

class XMLHandler(xml.sax.ContentHandler):
    def __init__(self):
        # Inicializamos variables de interés

    # Se llama cuando comienza un nuevo elemento
    def StartElement(self, tag, attributes):
        pass
    # Se llama cuando un elemento acaba
    def endElement(self, tag):
        pass

parser = xml.sax.make_parser()
parser.setFeature(xml.sax.handler.feature_namespaces, 0)
Handler = XMLHandler()
parser.setContentHandler(Handler)
parser.parse('models.xml') # nombre del documento a analizar
```

Cómo podríamos procesar con **`xml.sax`** este documento

```
<collection shelf="New Arrivals">
    <model number="ST001">
        <price>35000</price>
        <qty>12</qty>
        <company>Samsung</company>
    </model>
    <model number="RW345">
        <price>46500</price>
        <qty>14</qty>
        <company>Onida</company>
```

```
    </model>
</collection>
```

- Parser DOM

Librería **xml.dom**

```
# Procesar un determinado fichero
file = minidom.parse('model.xml')

# Obtener los elementos con un determinado tag
modelos = fil.getElementosByTagName('modelo')

# Obtener el atributo 'nombre' del segundo
    modelo
print('modelo #2 atributos:')
print(modelos[1].attributes['nombre'].value)

# El datos de un item específico
print('\nmodelo #2 datos:')
print(modelos[1].firstChild.data)
```

```
<data>
    <modelos>
        <modelo name='modelo1'>
            modelo1abc
        </modelo>
        <modelo name="modelo2">
            modelo2abc
        </modelo>
    </modelos>
</data>
```

- Librería BeatifulSoup

```
from bs4 import BeatifulSoup

# Leemos el fichero
with open('models.xml', 'r') as f:
    data = f.read()

# Pasamos los datos al parse
bs_data = BeatifulSoup(data, 'xml')

# Buscamos todas las instancias 'unique'
b_unique = bs_data.find_all('unique')
print(b_unique)

# Usamos .find búsquedas más concretas
b_name = bs_data.find('child', {'name':'Acer'})
print(b_name)
```

```
<modelo>
    <child name="Acer" qty="12">
        Portátil Acer
    </child>
    <unique>
        Número de modelo
    </unique>
    <child name="Acer" qty="7">
        Exclusive
    </child>
    <unique>
        1.200€
    </unique>
</modelo>
```

1.2.3) CSV (Comma-Separated Values)

- Formato basado en columnas normalmente separado por coma
- Sin embargo, el formato admite variaciones:
  - Con o sin cabecera con los nombres de las columnas
  - Separador de columans (comas, tabuladores, etc.)
  - Codificación de caracteres (UTF-8, latin-1, etc.)

- Escapado de caracteres (por ejemplo, una comilla doble como dos comillas dobles(" ") ó como \")
- Comillas opcionales (sólo si hacen falta) o en todos los campos siempre

- Carga en SQL

```
LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFLINE 'file_name'
    [RELACE | IGNORE]
    INTO TABLE tbl_name
    [PARCTITION (parctition_name, ...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMS}
        [TERMINATE BY 'string']
        [[OPTIONALL] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATE BY 'string']
    ]
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var, ...)]
    [SET col_name = expr, ...]
```

```
LOAD DATA LOCAL INFILE "/tmp/Posts.csv"
    INTO TABLE Posts
    COLUMNAS TERMINATED BY ',' ENCLOSED BY '"' ESCAPED BY '"'
    LINES TERMINATED BY '\r\n'
    IGNORE 1 LINES;
```

- Programación: Lectura con `Pandas DataFrame`