

# Bases de Datos II

Francisco Javier Mercader Martínez

## Índice

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Recuperación de datos y formatos de serialización</b> | <b>1</b> |
| 1.1      | Necesidad de formatos de serialización . . . . .         | 1        |
| 1.2      | Características a analizar . . . . .                     | 1        |
| 1.2.1    | Pandas . . . . .   | 1        |
| 1.2.2    | XML (eXtensible Markup Language) . . . . .               | 1        |
| 1.2.3    | CSV (Comma-Separated Values) . . . . .                   | 4        |

# Tema 1: Recuperación de datos y formatos de serialización

## 1.1) Necesidad de formatos de serialización

- Los formatos de [serialización](#)

## 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

```
1 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     <EDITORIA>Virgin</EDITORIA>
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 #REQUIRED>
3 <!ATTLIST DISCO TIPO=(CD | LP | DVD) "CD">
4 <!ELEMENT TITULO (#PCDATA)>
5 ...
6 <!ELEMENT REFERENCIA (EDITORIA, AÑO_EDICION) >
7 <!ELEMENT MUSICOS (MUSICO*)>
8 ...

```

Describe los documentos XML  $\Rightarrow$  [Validación](#)

#### • DTD (ii)

Documento XML:

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

```

1 <!DOCTYPE web-app
2   PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
3   "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`

```
1 import xml.sax
2
3 class XMLHandler(xml.sax.ContentHandler):
4     def __init__(self):
5         # Inicializamos variables de interés
6
7         # Se llama cuando comienza un nuevo elemento
8     def StartElement(self, tag, attributes):
9         pass
10
11    # Se llama cuando un elemento acaba
12    def endElement(self, tag):
13        pass
14
15 parser = xml.sax.make_parser()
16 parser.setFeature(xml.sax.handler.feature_namespaces, 0)
17 Handler = XMLHandler()
18 parser.setContentHandler(Handler)
19 parser.parse('models.xml') # nombre del documento a analizar
```

Cómo podríamos procesar con `xml.sax` este documento

```
1 <collection shelf="New Arrivals">
2     <model number="ST001">
3         <price>35000</price>
4         <qty>12</qty>
5         <company>Samsung</company>
6     </model>
7     <model number="RW345">
8         <price>46500</price>
9         <qty>14</qty>
10        <company>Onida</company>
11    </model>
12</collection>
```

- Parser DOM

Librería `xml.dom`

```

1 # Procesar un determinado fichero
2 file = minidom.parse('model.xml')
3
4 # Obtener los elementos con un determinado tag
5 modelos = fil.getElementsByTagName('modelo')
6
7 # Obtener el atributo 'nombre' del segundo
  modelo
8 print('modelo #2 atributos:')
9 print(modelos[1].attributes['nombre'].value)
10
11 # El datos de un item específico
12 print('\nmodelo #2 datos:')
13 print(modelos[1].firstChild.data)

```

```

1 <data>
2     <modelos>
3         <modelo name='modelo1'>
4             modelo1abc
5         </modelo>
6         <modelo name="modelo2">
7             modelo2abc
8         </modelo>
9     </modelos>
10 </data>

```

#### • Librería BeautifulSoup

```

1 from bs4 import BeautifulSoup
2
3 # Leemos el fichero
4 with open('models.xml', 'r') as f:
5     data = f.read()
6
7 # Pasamos los datos al parse
8 bs_data = BeautifulSoup(data, 'xml')
9
10 # Buscamos todas las instancias 'unique'
11 b_unique = bs_data.find_all('unique')
12 print(b_unique)
13
14 # Usamos .find búsquedas más concretas
15 b_name = bs_data.find('child', {'name': 'Acer'})
16 print(b_name)

```

```

1 <modelo>
2     <child name="Acer" qty="12">
3         Portátil Acer
4     </child>
5     <unique>
6         Número de modelo
7     </unique>
8     <child name="Acer" qty="7">
9         Exclusive
10    </child>
11    <unique>
12        1.200€
13    </unique>
14 </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 columnas (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

```

1 LOAD DATA [LOW_PRIORITY | CONCURRENT] [LOCAL] INFILE 'file_name'

```

```
2      [RELACE | IGNORE]
3      INTO TABLE tbl_name
4      [PARTITION (parctition_name, ...)]
5      [CHARACTER SET charset_name]
6      [{FIELDS | COLUMNS}
7          [TERMINATE BY 'string']
8          [[OPTIONALL] ENCLOSED BY 'char']
9          [ESCAPED BY 'char']
10     ]
11     [LINES
12         [STARTING BY 'string']
13         [TERMINATE BY 'string']
14     ]
15     [IGNORE number {LINES | ROWS}]
16     [(col_name_or_user_var, ...)]
17     [SET col_name = expr, ...]
```