Tema 6 - Formatos de representación de datos

David Moreno Lumbreras & Daniela Patricia Feversani
GSyC, EIF. URJC.

Laboratorio de Bases de Datos (BBDD)

Curso 2024-2025

Contenidos

6.1 Formatos de representación

Introducción a los formatos de datos

- **¿Qué son los formatos de datos?**: Los formatos de datos son las convenciones utilizadas para representar datos. Estos formatos pueden variar en términos de estructura, esquema y flexibilidad.
- **Importancia de los formatos de datos**: Los formatos de datos son fundamentales en la gestión de bases de datos ya que determinan cómo se almacenan, se organizan y se accede a los datos. Elegir el formato de datos adecuado puede tener un impacto significativo en el rendimiento de una base de datos.
- **Tipos de formatos de datos**: Los formatos de datos se pueden clasificar en tres categorías principales: datos estructurados, datos semi-estructurados y datos no estructurados. Cada uno tiene sus propias características y usos.

Tipos de datos

- **Datos estructurados**: Están sujetos a un **esquema fijo** de representación, conocido de antemano. Los datos se debe ajustar obligatoriamente a ese esquema.
  - Ejemplos: base de datos relacional, CSV ('Comma-Separated Values')/TSV('Tab-Separated Values').
  - A veces aparece el término datos tabulados como sinónimo.
- **Datos semi-estructurados**: Están sujetos a un **esquema flexible** de representación. Se definen los campos que almacenan la información y el tipo de dato de cada campo, así como atributos que ofrecen metadatos adicionales. Puede que falte algún campo y/o atributo (aunque se puede forzar a que se cumpla el esquema estrictamente).
  - Ejemplos: XML (Extensible Markup Language), JSON/JSONB (JavaScript Object Notation).
- **Datos no estructurados**: Se pueden interpretar siguiendo unas conjunto de reglas, pero la estructura y organización de su contenido es libre y no se conoce de antemano.
  - Ejemplo: texto en lenguaje natural (Documentos, correos electrónicos).

Datos en RDBMS

- Las bases de datos relacionales surgieron, originalmente, para almacenar datos estructurados (esquema fijo). Una vez acordados los campos de cada tabla y el tipo de dato de cada uno, no se puede cambiar.
- Si se cambia el esquema de una tabla, entonces hay que reconstruirla por completo.
- Posteriormente, muchas bases de datos relacionales han evolucionado para poder almacenar datos semi-estructurados o incluso no estructurados (con ciertas limitaciones). Por ejemplo:
  - PostgreSQL: `https://www.postgresql.org/docs/current/datatype.html`. Incluye soporte para datos semi-estructurados, como JSON o XML.
  - Oracle: Incluye soporte para datos semi-estructurados, como XML o JSON. Además, en sus versiones más recientes, ofrece extensiones para realizar búsquedas de texto en grandes bloques de caracteres (CLOB), es decir, datos no estructurados.

Formatos de representación de datos

Además de los tipos de datos especiales incluidos en PostgreSQL (geométricos, direcciones IP, etc.), existen otros estándares de representación de datos:

- CSV / TSV.
- JSON / JSONB (JSON binario).
- YAML.
- XML.
- Datos GIS (Geographical Information Systems).
- Formatos en sistemas distribuidos: Apache Parquet, Apache Avro, Apache Arrow ...

6.2 Datos CSV

Comma-separated values (CSV)

Formato clásico de representación de datos estructurados.

- Es un formato altamente flexible que permite ajustarse a las necesidades particulares de los usuarios o las aplicaciones.
- Se pueden definir múltiples *dialectos*: carácter separador de items en una fila, carácter delimitador de *strings*, carácter separador de decimales, formas de escapar caracteres especiales, etc. No es, ni mucho menos, universal.
- Todavía hoy día supone, en muchos casos, la forma más rápida de volcar o cargar datos en un sistema de gestión de datos estructurados (e.g. una base de datos relacional).
- Es un formato estandarizado (RFC 4180), aunque en la práctica no siempre se emplea.
- Cada **registro** (fila) contiene el mismo número de campos, separador por delimitadores. Deben aparecer siempre en el mismo orden. Si falta el valor para algún campo se deja el hueco entre dos delimitadores (*datos faltantes*).

6.3 Gestión de datos JSON

JavaScript Object Notation (JSON)

Formato estándar de representación de datos semi-estructurados en la web.

- No tienen un esquema definido, sino que se organizan mediante etiquetas que permiten agruparlos y crear jerarquías.
- Es un formato de inercambio de datos ligero y fácilmente legible por humanos.
- Su sintaxis está inspirada en la notación de objetos de JavaScript, pero es un formato independiente del lenguaje.

JavaScript Object Notation (JSON)

- Estándar abierto: `https://json.org/`, ECMA-404.
- Versión actual: RFC 8259.
- Se representan los datos como pares clave-valor.
  - Tipos de datos: numérico, *string*, booleano, array, objeto.
  - Un objeto se delimita por `{}`, un array se delimita con `[]`.
  - Separador de nombres con `:` y separador de valores con `,`.

JSON: ejemplo

```
{
"firstName": "John",
"lastName": "Smith",
"isAlive": true,
"age": 27,
"address": {
"streetAddress": "21 2nd Street",
"city": "New York",
"state": "NY",
"postalCode": "10021-3100"
},
"phoneNumbers": [
{
"type": "home",
"number": "212 555-1234"
},
{
"type": "office",
"number": "646 555-4567"
}
],
"children": [],
"spouse": null
}
```

JSON: JSON Schema

- **JSON Schema**: Herramienta empleada para validar estructuras de datos JSON.
- Consiste en un vocabulario que se puede emplear para establecer consistencia, validez e interoperabilidad a los datos JSON.
- Establece un lenguaje común para intercambiar datos, definiendo reglas de validación precisas para las estructuras de datos.
- Un esquema JSON define las reglas que deben seguir los datos JSON. Se emplean validadores de esquemas para comprobar que los datos cumplen con el esquema.
- JSON Schema está estandarizado (Versión 2020-12). JSON Schema.

JSON: JSON Schema

```
{
"$schema": "http://json-schema.org/draft-07/schema#",
"title": "Persona",
"type": "object",
"properties": {
        "nombre": {
        "type": "string"
        },
        "edad": {
        "type": "integer",
        "minimum": 0
        },
        "ciudad": {
        "type": "string"
        }
},
"required": ["nombre", "edad"]
}
```

```
{
"nombre": "Juan",
"edad": 30,
"ciudad": "Madrid"
}
```

```
{
"nombre": "Juan",
"ciudad": "Madrid"
}
```

JSON en PostgreSQL

- Los formatos de datos JSON y JSONB están soportados desde la versión 9.4.
- PostgreSQL tiene dos tipos de datos para almacenar información JSON: **JSON** y **JSONB**.
  - **JSON**: Almacena una copia exacta del texto de entrada JSON. Incluye espacios en blanco o el orden de las claves, pero no permite la indexación.
  - **JSONB**: Almacena datos JSON en un formato binario descompuesto. Esto siginifica que se almacenan de una manera que permite una búsqueda y manipulación más eficientes.
    - No conserva espacios en blanco, ni orden de las claves, ni claves duplicadas.
    - Permite la indexación.
    - Se recomendia usar JSONB debido a su eficiencia.
- JSON Types.
- JSON functions and operators.

6.4 Gestión de datos XML

eXtensible Markup Language (XML)

- Metalenguaje (lenguaje para describir otros lenguajes), que permite a los diseñadores crear sus propias etiquetas personalizadas para proporcionar funcionalidad no disponible en HTML.
- XML fue diseñado para almacenar y transportar datos.
- Es autodescriptivo, es decir, las etiquetas definidas son las encargadas de proporcionar un contexto a los datos.
- Es un estándar abierto, simple, extensible, reutilizable, que separa contenido de la presentación del mismo.
- Almacena datos en un formato de texto plano, lo que propociona una forma independiente de Software y Hardware para almacenar, transportar y compartir datos.
- Soporte en bases de datos relacionales: estándar SQL/XML (Desde el estándar 2003 y 2006).

Estructura documento XML

- Los datos XML presentan una estructura jerárquica formada por un conjunto de elementos correctamente anidados que no se solapan entre ellos.
  - **Elemento Raíz o Root**: Es un elemento único en el XML que se encarga de contener a todos los demás.
  - **Prólogo o Declaración**: sección inicial, opcional. Indica la versión de XML, codificación y si se debe chequear contra una DTD. También puede indicar enlaces a *stylesheet* y documento DTD.
  - **Etiquetas**: Construcción de marcado que comienza con $<$ y termina con $>$ y que, distingue entre mayúsculas y minúsculas. Cada elemento de un documento XML debe estar delimitado por una etiqueta de inidcio y una etiqueta de fin.
  - **Entidades**: XML ofrece métodos (entidades) para referir a algunos caracteres especiales reservados.
    - $<$ → <
    - $>$ → >
    - & → &
    - ' → '
    - " → "

XML: ejemplo

```
<?xml version= "1.1" encoding= "UTF-8" standalone= "no"?>
<?xml:stylesheet type = "text/xsl" href = "staff_list.xsl"?>
<!DOCTYPE STAFFLIST SYSTEM "staff_list.dtd">
<STAFFLIST>
        <STAFF branchNo = "B005">
                <STAFFNO>SL21</STAFFNO>
                        <NAME>
                                <FNAME>John</FNAME><LNAME>White</LNAME>
                        </NAME>
        <POSITION>Manager</POSITION>
        <DOB>1945-10-01</DOB>
        <SALARY>30000</SALARY>
        </STAFF>
        <STAFF branchNo = "B003">
                <STAFFNO>SG37</STAFFNO>
                ...
</STAFFLIST>
```

XML: Definición de Tipo de Documento

- Para ser válido, un documento XML necesita cumplir ciertas reglas de semántica que se definen en un esquema XML o en una Definición de Tipo de Documento (*DTD*).
- El documento DTD indica el formato y número de ocurrencias que pueden aparecer de un elemento, siendo ocurrencias el número de veces que un elemento específico puede aparacer en el XML:
  - Un * indica cero o más ocurrencias de un elemento.
  - Un + indica una o más ocurrencias de un elemento.
  - Un ? indica cero o *exactamente* una ocurrencia.
  - Cualquier elemento sin cualificador debe ocurrir exactamente una vez.
  - `#PCDATA` indica datos en formato caracter parseable.
  - `CDATA` indica que se pase el texto delimitado directamente a la aplicación, sin interpretación.

DTD: ejemplo

```
<!- DTD para el documento XML del ejemplo previo ->

<!ELEMENT STAFFLIST (STAFF)*>
<!ELEMENT STAFF (NAME, POSITION, DOB?, SALARY)>
<!ELEMENT NAME (FNAME, LNAME)>
<!ELEMENT FNAME (#PCDATA)>
<!ELEMENT LNAME (#PCDATA)>
<!ELEMENT POSITION (#PCDATA)>
<!ELEMENT DOB (#PCDATA)>
<!ELEMENT SALARY (#PCDATA)>
<!ATTLIST STAFF branchNo CDATA #IMPLIED>
```

XML: otras tecnologías

- Existen un conjunto de tecnologáis y estándares que permiten trabajar con documentos XML.
  - Interfaces (APIs): DOM (basada en árbol, vista orientada a objetos) y SAX (basada en eventos, acceso serializado). Permiten a los programas interactuar con documentos XML.
  - *Namespaces*: Nombres de elementos y relaciones cualificados, para evitar colisiones en elementos con mismo nombre y definiciones distintas.
  - XSLT: Mecanismos para transformación de XML en otros formatos, como HTML o SQL.
  - XPath: Lenguaje de consulta declarativo para recuperar elementos de un documento XML.
  - XPointer: Proporciona acceso a los valores de atributos o el contenido de elementos en cualquier lugar de un documento XML.
  - XML Schema: Permite definir la estructura de un documento XML de forma más robusta que con DTD. El esquema está escrito en XML.

Soporte XML en PostgreSQL

Resumen: PostgreSQL vs SQL/XML Standards.

- PostgreSQL incluye el tipo de dato XML.
- También incluye una lista de funciones para XML, conformes con el estándar SQL/XML.
- Igualmente, incluye funciones para mapear una tabla, esquema o catálogo de la base de datos a un documento XML y un XML Schema.
- Incluye, también, varias características adicionales y funciones, más allá de las definidas en el estándar.

```
SELECT xpath('/libro/titulo/text()', contenido_xml)
FROM biblioteca
WHERE id = 1;
```

6.5 Gestión de datos geográficos

Sistema de Información Geográfica (GIS)

- Un Sistema de Información Geográfica (GIS en inglés), debe implementar una serie de características y servicios para manipulación de datos geográficos:
  - Lectura, edición, almacenamiento y gestión (en general), de datos espaciales.
  - Análisis de esos datos, incluyendo consultas sencillas o elaboración de modelos complejos, tanto sobre la componente espacial de los datos (localización de cada valor o elemento) como sobre la componente temática (valor o elemento en sí).
  - Generación de resultados: informes, gráficos, mapas, etc.
- En su forma básica, SQL no recoge las goemetrías que forman la parte espacial de una entidad. Para estandarizar, Simple Features for SQL (SFS) define tipos estandarizados para geometrías, basados en OpenGIS Geometry Model.

Libro en español: "Sistemas de Información Geográfica" - V. Olaya. (CC-BY, 2020).

PostGIS

- PostGIS es una extensión para PostgreSQL que añade soporte para objetos geográficos, permitiendo consultas de localización en formato SQL.
- https://postgis.net/.
- También se incluyen muchas otras características que, según la página del proyecto, están raramente presentes en otras bases de datos espaciales (como Oracle Locator/Spatial y SQL Server).
  - Listado completo de características.
- Manual completo (v3.1).

PostGIS

- Incluye el tipo `Geography`, con soporte nativo para características espaciales representadas por coordenadas geográficas (lat/lon). Son coordenadas esféricas expresadas en unidades angulares.
- Posteriormente, se pueden efectuar consultas SQL para recuperar los datos espaciales.
- Igualmente, también se pueden volcar los datos a un archivo *shape*, mediante la herramienta `pgsql2shp`.
- Algunos índices de PostgreSQL están específicamente pensados para trabajar con datos espaciales:
  - GiST.
  - BRIN.
  - SP-GiST.

6.6 Otros formatos de representación de datos

Apache Parquet

- https://parquet.apache.org/.
- Formato de almacenamiento de datos *columnar*, que incluye una serie de optimizaciones especialmente diseñadas para cargas de trabajo de análisis de datos.
- Comprime los datos de columnas, ahorrando espacio de almacenamiento y permitiendo lectura de columnas individuales en lugar de archivos completos.
- La lectura de datos en formato Parquet es, casi siempre, mucho más eficiente que en formato CSV o JSON.
- También soporta tipos de datos complejos: array, *map* o *struct*

Apache Avro

- https://avro.apache.org/.
- Formato de serialización de datos, frecuentemente ligado a Apache Hadoop.
- Incluye un framework para RPCs y formato de serialización orientado a *filas*.
- El formato de serialización permite tanto almacenamiento persistente de datos como envío a través del cable entre nodos de un cluster.
- El esquema de definición de la estructura de datos se puede realizar en JSON.
- Es similar a otras alternativas como Thrift y Protocols Buffers.

Apache Arrow

- https://arrow.apache.org/.
- Formato de representación de datos columnar en memoria, independiente de lenguaje.
- Soporta datos planos o jerárquicos y está optimizado para CPUs y GPUs modernas.
- Soporta *zero-copy reads*: el procesador no participa en la copia de datos de un área de memoria a otra.
- Librerías disponibles en múltiples lenguajes.
- Arrow vs Parquet.

Bibliografía I


[Connolly & Begg, 2015] Connolly, T., Begg, C.
Database Systems.
Pearson, 6th Global Edition. 2015.


[Petrov, 2019] Petrov, A.
Database Internals.
O'Reilly Media, 2019.