Bancos de Dados Semi-Estruturados

Gabriel Pordeus Santos
Gabriel Simonetto
Gustavo Kundlatsch

30 de março de 2021

Resumo

Neste trabalho, feito para a matéria de Bancos de Dados II, do curso de Ciências da Computação da Universidade Federal de Santa Catarina, apresentaremos a diferença entre dados tradicionais e dados semi-estruturados, o que eles são, para que servem e quais suas implicações. Para explicar o tema, serão utilizados dois exemplos: XML e JSON.

1 Dados Semi-Estruturados

Podemos dividir um banco de dados tradicional em duas partes: seu esquema e seus dados. O esquema é todo o esqueleto do banco, que descreve as tabelas, as chaves e todo o resto da estruturação, enquanto os dados são as entradas armazenadas. Todavia, existem dados cuja natureza *não é* estruturada. Dados não estruturados são aqueles no qual a estrutura está contida dentro dos próprios dados [1]. O principal exemplo de dados semi-estruturados é a própria internet, pois os dados não seguem necessariamente um padrão, e podem ter diversos níveis de organização diferentes. Outros exemplos menos diretos são binários executáveis, pacotes TCP/IP e arquivos compactados [2]. Uma definição simples e direta de dado semi-estruturado é: dados que de um ponto de vista prático não são nem crus nem estritamente tipado através de tabelas ou grafos [3].

Existem diversas características que diferem os dados semi-estruturados dos dados organizados da maneira clássica [4]. Esse tipo de dado normalmente possui a estrutura definida após eles já existirem, ou seja, apesar de nem sempre existir algum esquema implícito, é possível extrair alguma organização dos dados apresentados. Apesar disso, a estrutura é irregular, pois dados similares não necessariamente respeitam a mesma organização. Essa organização, que pode ser implícita ou explícita, as vezes compreende apenas alguma parte do dado, uma vez que não existem restrições para a estruturação (nada obriga que todo o documento respeite as classificações definidas). Além disso, a estrutura é extensa, uma vez que os dados são heterogêneos, e evolucionária, pois seus valores mudam constantemente (especialmente no caso da web). Por fim, uma de

Figura 1: Distinção entre dados estruturados, semi-estruturados e não estruturados. Fonte: https://universidadedatecnologia.com.br/dados-estruturados-e-nao-estruturados/

Dados tradicionais	Dados semi-estruturados
Esquema predefinido	Nem sempre há um esquema predefinido
Estrutura regular	Estrutura irregular
Estrutura independente dos dados	Estrutura embutida no dado
Estrutura reduzida	Estrutura extensa
Estrutura fracamente evolutiva	Estrutura fortemente evolutiva
Estrutura prescritiva	Estrutura descritiva
Distinção entre estrutura e dado é clara	Distinção entre estrutura e dado não é clara

Tabela 1: Diferenças entre dados tradicionais e dados semi-estruturados [4].

suas características mais marcantes é que os dados se misturam com a estrutura, e não é possível ter uma distinção clara desses dois elementos.

A tabela 1, retirada do artigo de Mello et. al. [4] expõem as diferenças entre os bancos de dados semi-estruturados e os bancos de dados estruturados.

O uso de dados semi-estruturados permite que os desenvolvedores integrem diversos sistemas e integrem diversos fluxos de dados, uma vez que a estrutura desse tipo de dado é flexível. Como sites e APIs podem evoluir rapidamente com o tempo, o uso de dados semi-estruturados garante que a integração seja contínua, pois sua estrutura é evolutiva [5].

Apesar de ser uma ferramenta poderosa, os dados semi-estruturados possuem vantagens e desvantagens em relação aos dados estruturados. As principais vantagens já foram citadas, como a capacidade de suportar diversas fontes de dados e a evolução de seus formatos. Já entre as desvantagens, podemos citar a dificuldade que existe para realizar consultas de maneira eficiente, tanto que

consultas em bancos semi-estruturados são uma linha de pesquisa por si só (ao contrário de dados tradicionais, que possuem o SQL como linguagem padrão bem implementada e otimizada). Além disso, como as restrições do esquema não existem para os dados semi-estruturados, o banco está mais propenso a receber dados corrompidos, incorretos e todo o tipo de lixo. Portanto, é necessário ter cuidado ao escolher utilizar esse tipo de banco de dados, e manter a atenção para esses detalhes durante a implementação do aplicativo que utilizará a base.

Para exemplificar os bancos de dados semi-estruturados e demonstrar seu uso prático, apresentaremos duas linguagens capazes de descrever dados dessa maneira, o XML e o JSON.

2 XML

```
<?xml version="1.0" standalone="yes" ?>
- <shop location="Birmingham" size="Large">
  - <food>
      <Name>Apple</Name>
      <type>fruit</type>
      <cost>15</cost>
    </food>
  - <food>
      <Name>Carrot</Name>
      <type>vegetable</type>
      <cost>10</cost>
    </food>
  </shop>
```

Figura 2: Exemplo de arquivo XML.
Fonte: https://www.jonathanmedd.net/2009/12/powershell-2-0-one-cmdlet-at-a-time-21-select-xml.html

2.1 O que é?

O XML (Extensible Markup Language) é uma linguagem de marcação responsável por descrever classes de dados para ampla distribuição na internet, focada em ser legível tanto por humanos quanto por maquinas, e em permitir a interoperabilidade entre diferentes sistemas e aplicações[6].

Uma das características centrais do XML é que suas tags (marcadores semânticos) nao são pré-definidos, isso significa que fica aberto para o usuário de XML descrever sua estrutura de dados da melhor forma que lhe convenha [7]. A união dessa liberdade semântica com a padronização da sintaxe inerente ao XML o torna uma ferramenta eficaz em distribuir dados na internet.

Existem 3 possíveis estados para um documento XML[8]:

- Inválido: Não segue as regras de sintaxe de XML. Caso um documento XML nao siga as regras semânticas definidas no DTD (Document Type Definition) ou no schema definido pelo desenvolvedor isso também torna inválido.
- Well formed: Segue as regras de sintaxe do XML, mas não possui regras semanticas definidas num DTD ou schema.
- Válido: Segue regras sintáticas e semânticas.

2.2 Well-formedness - A Sintaxe do XML

Uma das características marcantes do XML em relação a outras linguagens de marcação (HTML, usado para mostrar conteúdo em paginas web, e seu ancestral direto, o SGML), é a de que a leitura de um documento XML deve ser não ambígua, isso garante que não ocorrerão erros durante a leitura do XML, para isso, definem-se uma coleção de regras que tem como objetivo garantir essa facilidade de leitura do documento XML [9].

Alguns exemplos dessas regras são:

- Posicionamento da declaração de XML: um header inicial que define metadados para a leitura do arquivo, é opcional, mas caso esteja presente, precisa estar na primeira linha do arquivo.
- Presença de um elemento raiz: o elemento raiz deve encapsular todos os outros elementos.
- Elementos não podem se intercalar: um elemento contem outro completamente, tanto sua abertura quanto fechamento.
- XML é case sensitive: não é possível fechar uma tag minuscula em maiúsculo.

Demonstração:

```
<!-- Esta é a sintaxe de comentários em xml -->

<!-- Declaração de XML -->
<?xml version="1.0" encoding="ISO-8859-1" standalone="no"?>

<!-- Exemplo de quebra do elemento raiz -->
<section_1>
  Hello, World!
</section_1>
<section_2>
  <!-- Exemplo de elementos intercalando-se indevidamente -->
  <p>
    <b>I
```

```
14      <i>really love
15      </b> XML.
16      </i>
17    </p>
18
19    <!-- Exemplo de tag quebrando a regra de case sensitive -->
20    <h1>Elements are case sensitive</H1>
21  </section_2>
```

2.3 DTD e Schema - A Semântica do XML

DTD (Document Type Definition) consiste de uma gramática que estrutura de que forma as tags se relacionam entre si. Mencionamos anteriormente que uma das forças do XML é a liberdade do usuário em customizar o documento para seu caso de uso, entretanto, em meio a um mesmo caso de uso, é útil possuir constrições para garantir que diferentes fontes para um mesmo escopo estejam padronizadas: quais atributos podem ser aplicados a cada elemento, em que ordem podem aparecer, quais relações hierárquicas são permitidas[10].

Exemplo de um DTD Interno (linhas 2 até 7):

```
1   <?xml version="1.0"?>
2  <!DOCTYPE note [
3  <!ELEMENT note (to,from,heading,body)>
4  <!ELEMENT to (#PCDATA)>
5  <!ELEMENT from (#PCDATA)>
6  <!ELEMENT heading (#PCDATA)>
7  <!ELEMENT body (#PCDATA)>
8  ]>
9  <note>
10 <to>Tove</to>
11 <from>Jani</from>
12 <heading>Reminder</heading>
13 <body>Don't forget me this weekend</body>
14 </note>
```

Basicamente, definimos o doctype note, e o elemento note, que pode conter as tags (to,from,heading,body) internamente, e estas, todas usam o tipo primitivo PCDATA (significando *parsed character data*)

Acima, fomos apresentados ao DTD Interno, que possui esse nome por ser definido dentro do próprio XML que o usa, uma outra forma de definir o DTD seria apontar para um arquivo encontrado localmente[11]:

```
1  <!DOCTYPE note SYSTEM note.dtd>
```

Dentro de note.dtd possuiríamos os mesmos elementos que usamos no exemplo acima por exemplo. Também é possível usar DTD's instanciadas publicamente, muito como faríamos com a importação de bibliotecas em uma linguagem de programação[12]:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN">
```

Uma alternativa ao DTD é o Schema, que vai possuir a mesma contextualização, porém, será muito mais poderoso devido a algumas propriedades adicionais que possui sobre o DTD[13]:

- Uso de namespaces.
- Maior flexibilidade na definição de restrições com o uso de regex, ranges numéricos e enumeradores.
- Mais possibilidades para definição de campos unicos, ao contrário de ID e IDREF em DTDS, podem ser definidos em qualquer parte do documento, serem de qualquer tipo de dado, e podem se referir tanto a elementos quanto a conteúdo.

Outro fator interessante é que o schema é construido sobre uma sintaxe em XML! Tornando o uso mais legível e intuitivo:

```
<xs:element name="note">

<xs:complexType>
  <xs:sequence>
    <xs:element name="to" type="xs:string"/>
    <xs:element name="from" type="xs:string"/>
    <xs:element name="heading" type="xs:string"/>
    <xs:element name="body" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

</xs:element>
```

2.4 Quais as Implicações?

O XML é um padrão de definição de dados amplamente utilizado ao longo da internet, isso fez com que fosse possível que desenvolvedores pudessem transmitir informações críticas por meio de API's, ampliando a comunicação de serviços e permitindo a construção conjunta de soluções mais e mais complexas. Essa ampla utilização do XML fez com que diversas tecnologias fossem construídas em cima do XML, ja utilizando dos padrões definidos, tais como XHTML, MathML, SVG, XUL, XBL, RSS, e RDF[7].

Também um agrupamento de especificações formou-se em torno do XML para ampliar e especificar os usos de XML[10]:

- XPath (XML Path Language), uma linguagem para referenciação de componentes de um arquivo XML. O XPath é usado amplamente por outras especificação em XML e em bibliotecas de programação para busca de dados definidos em XML.

- XQuery (XML Query) é uma linguagem de consulta construida em cima de XPath e XML Schema que providencia métodos para acessar, manipular e retornar XML, o que a torna perfeita para uso em conjunto com bancos de dados em XML.

- XML Encryption que define uma sintaxe para criptografia de conteúdo em XML.

Concluindo, o advento do XML permitiu que a troca de dados pudesse ser simplificada, uma vez que foi possível que linguagens e programadores pudessem focar na construção do ecossistema ao construir ferramentas que pudessem interpretar e construir XML que pudesse ser amplamente distribuído. XML permite a construção de código e buscas eficientes, a presença de metadados contextualizantes em torno dos elementos de dados faz com que o programador encontre o que busca[8].

3 JSON

3.1 O que é?

JSON, um acrônimo para ***J**ava**S**cript **O**bject **N**otation*, é um formato leve para troca de dados baseado em um subconjunto do padrão ECMA-262, terceira edição (dezembro de 1999). Se propõe a ser fácil de ler e escrever para humanos e fácil de analisar e gerar para máquinas. É um formato de texto independente de linguagem, mas utiliza convenções familiares à programadores de linguagens de programação da família do C. [14].
Segundo o seu "descobridor", como Douglas Crockford se define, o JSON **não** é [15]:

- um formato de documento.

- uma linguagem de marcação

- uma linguagem de serialização geral, já que não possui suporte direto à grafos cíclicos, estruturas binárias ou funções.

3.2 Sintaxe

O JSON é construído em cima de duas estruturas: uma coleção de pares nome/valor e uma lista ordenada de valores, É definido como: [14]

- Um objeto JSON é delimitado por chaves (,) e composto por pares string/-valor, separados por vírgula (,).

- Uma string é uma cadeia de zero ou mais caracteres Unicode, delimitada por aspas duplas (") e usando a barra invertida (\) como escape para aspas e caracteres de controle.

- Um valor pode ser uma string, um número, um objeto, uma array, true, false ou null.
- Um número é uma cadeia de dígitos, também delimitada por aspas (”), contendo ou não um ponto (.) e mais dígitos à direita, fracionários. Pode conter, também parte exponencial com um e (e, E) seguido de sinal (+,-) e mais digitos.
- Uma array é uma lista ordenada de valores, delimitada por colchetes ([,]) e separada por vírgula (,).

Demonstração [16]:

```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Konfabulator Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
        "vOffset": 250,
        "alignment": "center"
    },
    "text": {
        "data": "Click Here",
        "size": 36,
        "style": "bold",
        "name": "text1",
        "hOffset": 250,
        "vOffset": 100,
        "alignment": "center",
        "onMouseUp": "sun1.opacity = (sun1.opacity / 100) * 90;"
    }
}}
```

Mesmo texto, em XML [16]:

```
<widget>
    <debug>on</debug>
    <window title="Sample Konfabulator Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>
```

```
7       </window>
8       <image src="Images/Sun.png" name="sun1">
9           <hOffset>250</hOffset>
10          <vOffset>250</vOffset>
11          <alignment>center</alignment>
12      </image>
13      <text data="Click Here" size="36" style="bold">
14          <name>text1</name>
15          <hOffset>250</hOffset>
16          <vOffset>100</vOffset>
17          <alignment>center</alignment>
18          <onMouseUp>
19              sun1.opacity = (sun1.opacity / 100) * 90;
20          </onMouseUp>
21      </text>
22  </widget>
```

3.3 Para que Serve?

JSON serve para a transmissão de dados entre quaisquer tipos de programas que precisem se comunicar, sendo o programador o responsável por manter a consistência semântica [15].

Mais especificamente, é especialmente útil em cenários de comunicação com navegadores ou aplicações mobile nativas. Pode ser útil em comunicação servidor-a-servidor ou ser deixado de lado em favor de frameworks de serialização [17].

No caso de bancos de dado NoSQL, é necessário trabalhar com os dados fornecidos no formato fornecido. Já bancos de dados relacionais são melhor adaptados a dados mais estruturados, com um schema específico[17],

3.4 Quais as Implicações?

Nos anos 2000, a interação com a web começou a se transformar. Na época, o navegador apenas exibia a informação que o servidor trabalhava. Quando se clicava em um link, um pedido era enviado ao servidor, que processava e devolvia a informação como HTML, recarregando a página. Esse processo é ineficiente, precisando-se alterar tudo mesmo quando as diferenças afetam apenas uma pequena porção da página [17].

Dentre as novas tecnologias para resolver esse problema, surge o Javascript, que eventualmente se torna a linguagem de programação universal nos navegadores [17], com excelente sintaxe para objetos e literais de array [18] e o JSON, como uma forma de comunicação leve por onde se dá essa comunicação e representação de tipos fundamentais de dados [18].

Outro exemplo de aplicação de JSON é a técnica AJAX (*Asynchronous Javascript and XML*), que apesar de inicialmente planejada para XML, pode muito bem ser utilizado com JSON, muitas vezes mais eficientemente [19]. Assim

como as APIs REST, que - apesar de poderem utilizar XML ou YAML, são mais comumente implementadas com JSON [20].

4 Conclusão

O objetivo desse trabalho foi apresentar os bancos de dados semi-estruturados, seu conceito, onde ele estão presentes e para que servem. Para isso, apresentamos dois estudos de caso, as linguagens XML e JSON, explicando a sintaxe de cada uma com exemplos de código, para que surgiram e como são utilizadas.

Hoje em dia os dados semi-estruturados estão presentes no dia-a-dia de todas as pessoas que utilizam a internet, seja através de navegadores que utilizam o HTML ou através de aplicativos (desktop e mobile) que fazem requisições a servidores utilizando o padrão REST. Por conta disso, as pesquisas em banco de dados semi-estruturados são de suma importância para a otimização de consultas e melhorias no design desse tipo de banco, uma vez que uma pequena melhora pode afetar milhões de aplicações, gerando uma experiência melhor para os usuários e economizando dados de armazenamento dos servidores e da transferência das informações.

Referências

[1] P. Buneman, "Semistructured data," in *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pp. 117–121, 1997.

[2] G. for Geeks, "What is semi-structured data?." `https://www.geeksforgeeks.org/what-is-semi-structured-data/`, Nov. 2020.

[3] S. Abiteboul, "Querying semi-structured data," in *International Conference on Database Theory*, pp. 1–18, Springer, 1997.

[4] R. dos Santos Mello, C. F. Dorneles, A. Kade, V. de Paula Braganholo, and C. A. Heuser, "Dados semi-estruturados,"

[5] T. H. T. Group, "What is semi-structured data? 5 key things to know." `https://treehousetechgroup.com/what-is-semi-structured-data-5-key-things-to-know/`, Nov. 2020.

[6] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, *et al.*, "Extensible markup language (xml) 1.0," 2000.

[7] Mozilla, "Xml introduction." `https://developer.mozilla.org/en-US/docs/Web/XML/XML_introduction/`.

[8] IBM, "Introduction to xml." `https://www.ibm.com/developerworks/xml/tutorials/xmlintro/xmlintro.html`.

[9] Computerphile, "Sgml html xml what's the difference? (part 1) - computerphile." `https://www.youtube.com/watch?v=RH0o-QjnwDg`.

[10] Wikipedia, "Xml." `https://en.wikipedia.org/wiki/XML`.

[11] Wikipedia, "Document type definition." `https://en.wikipedia.org/wiki/Document_type_definition`.

[12] W3, "Html 4.01 transitional dtd." `https://www.w3.org/TR/html4/loose.dtd`.

[13] Wikipedia, "Xml schema." `https://en.wikipedia.org/wiki/XML_schema`.

[14] D. Crockford, "Introducing json." `https://www.json.org/json-en.html`.

[15] D. Crockford, "The history of json." `https://youtu.be/TjVcVWB0oFk`, 2019.

[16] "Json example." `https://www.json.org/example.html`.

[17] J. Freeman, "What is json? a better format for data exchange," *InfoWorld*, 2019.

[18] S. Willison, "Why json isn't just for javascript," *Simon Willison's Weblog*, 2006.

[19] A. T. H. III, *Ajax: The Definitive Guide*. O'Reilly Media, Inc., 2008.

[20] "An introduction to rest and json." `https://youtu.be/u-RnFs9dby4/`, 2017.