MAC0439
Laboratório de Bancos de Dados

Dados Semiestruturados

Introdução a JSON (JavaScript Object Notation)

Profª. Kelly Rosa Braghetto
DCC-IME-USP

16 de março de 2020

JavaScript Object Notation (JSON)

- É um padrão para a seriação de objetos de dados (geralmente em arquivos)
- Devido ao seu formato “leve”, é útil também para:
  - Trocar dados
  - Representar e armazenar dados semiestruturados
- Assim como XML, é independente de linguagem de programação
  - JSON usa a sintaxe de JavaScript para descrever objetos de dados, mas os *parsers* e bibliotecas para JSON existem em várias linguagens de programação diferentes

Outras características

- Assim como XML: JSON é hierárquico, autodescritivo e fácil de entender
- Mas tem vantagens sobre a XML:
  - JSON permite definir vetores de elementos
  - JSON é menor, mais fácil de escrever e mais fácil de processar
    - O menor tamanho dos documentos JSON é devido principalmente à ausência de marcações de fim de elementos

JSON – Exemplo (arquivo “livros.json”)

```
{ "Livros":
  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Preço":85,
      "Edição":3,
      "Título":"A First Course in Database Systems",
      "Autores":[ {"Nome":"Jeffrey", "Sobrenome":"Ullman"},
                  {"Nome":"Jennifer", "Sobrenome":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Preço":100,
      "Nota":"Compre também o livro 'A First Course' e faça um excelente negócio!",
      "Título":"Database Systems: The Complete Livro",
      "Autores":[ {"Nome":"Hector", "Sobrenome":"Garcia-Molina"},
                  {"Nome":"Jeffrey", "Sobrenome":"Ullman"},
                  {"Nome":"Jennifer", "Sobrenome":"Widom"} ] }
  ],
  "Revistas":
  [
    { "Título":"National Geographic",
      "Mês":"Janeiro",
      "Ano":2009 }
    ,
    { "Título":"Newsweek",
      "Mês":"Fevereiro",
      "Ano":2009 }
  ]
}
```

Pares nome:valor

- Dados em JSON são escritos como pares do tipo **nome:valor**
- Um par consiste em um nome de campo (delimitado por aspas*), seguido por dois-pontos (':') e um valor

“nome”:“Ana”

- Isso é fácil de entender; em uma linguagem de programação, equivale ao comando:

nome = “Ana”

(*) Em JavaScript, diferentemente, os nomes nos pares podem ser também números ou identificadores, como em: {nome: “Ana”}

Construtores básicos

Um **valor** pode ser um valor simples, um objeto ou um vetor

- **Valores simples** – são valores do tipo
  - ***string*** (delimitados por aspas),
  - **número** (inteiro ou em ponto flutuante) , ou
  - **boolean**
- **Objetos** – são conjuntos de pares do tipo **nome:valor**, separados por vírgula e delimitados por chaves ('{' e '}')
- **Vetores** – são listas ordenadas de valores separados por vírgulas e delimitados por colchetes ('[' e ']')
- Valores simples predefinidos em JSON: **null**, **true** e **false**

Objetos

- A definição de um objeto é delimitada por '{' e '}'
- Um objeto pode conter vários pares nome:valor

```
{“nome”:“Ana”,“sobrenome”:”Gomes”}
```

- Equivale a:

```
nome = “Ana”
sobrenome = “Gomes”
```

Vetores

- Vetores em JSON são delimitados por '[' e ']'
- Um vetor pode conter vários valores

```
{"empregados":[
    {"nome":"Paulo","sobrenome":"Costa"},
    {"nome":"Ana","sobrenome":"Gomes"},
    {"nome":"Pedro","sobrenome":"Neves"}]}
```

JSON

- Podemos ter:
  - **Vetores vazios**. Ex.:

```
{“Autores”:[ ]}
```

  - **Vetores que contêm valores simples e objetos aos mesmo tempo.** EX.:

```
{“Autores”:[ {"Nome":"Jeffrey","Sobrenome":"Ullman"},
"Jennifer Widom" ]}
```

- Objetos devem sempre ser compostos por pares nome:valor
  - Exemplo inválido:

```
{“Autores”:[ {"Nome":"Jeffrey","Sobrenome":"Ullman"},
             {"Jennifer”, “Widom"} ]}
```

JSON, JavaScript e AJAX (Um parênteses)

- Uso comum de JSON:
  - obter dados em JSON de um servidor web (como um arquivo ou um HttpRequest),
  - convertê-los em objetos JavaScript
  - usar os objetos em um página Web
- Um programa em JavaScript pode usar a função embutida ***eval()*** ou a função ***JSON.parse()*** para "processar" dados em JSON e produzir objetos JavaScript com um único comando
  - Comparação: para obter objetos a partir de dados em um documento XML, seria preciso usar uma API como a XML DOM para percorrer o documento, extrair valores e depois atribuí-los a variáveis ou objetos
- Devido a isso, desenvolvedores Web que usam AJAX rapidamente aderiram ao uso do JSON
- Curiosidade: AJAX = *Asynchronous JavaScript and XML*

Sobre o AJAX (Continuando o parênteses)

- AJAX é um conjunto de tecnologias de desenvolvimento Web, suportadas por navegadores, para tornar páginas Web mais interativas com o usuário, por meio da execução de tarefas do lado do cliente.
- Com AJAX é possível:
  - atualizar uma página sem recarregá-la
  - requisitar dados de um servidor depois da página ter sido carregada
  - receber dados de um servidor depois da página ter sido carregada
  - enviar dados a um servidor em *background*

De JSON para JavaScript (Continuando o parênteses)

- Como a sintaxe de JSON é um subconjunto da sintaxe da JavaScript, a função ***eval()*** de JavaScript pode ser usada para converter um texto em JSON para um objeto em JavaScript
  - A função ***eval()*** executa um código em JavaScript contido na *string* passada como parâmetro para a função
  - ***eval()*** usa o interpretador da JavaScript, que fará o processamento do texto JSON e produzirá o objeto em JavaScript
- A função ***JSON.parse()*** é mais segura!
  - Evita a execução de código malicioso

De JavaScript para JSON (Continuando o parênteses)

- É possível gerar uma *string* a partir de (= “seriar”) um objeto JSON em JavaScript por meio da função ***stringify()***. Exemplo:

```
<script>
var Autor1 = new Object();
var Autor2 = new Object();
var Livro = new Object();

Autor1.nome = "Jeffrey";
Autor1.sobrenome = "Ullman";

Autor2.nome_completo = "Jennifer Widom";
Autor2.afiliacao = "Universidade de Stantord"

Livro.titulo = "Database Systems - The Complete Book";
Livro.autores = [Autor1, Autor2, "Hector Garcia-Molina"]

// “seria” o objeto usando JSON.stringify() e exibe o resultado
console.log(JSON.stringify(Livro));
</script>
```

Resultado:

{"título":"Database Systems - The Complete Book","autores":[{"nome":"Jeffrey","sobrenome":"Ullman"},
{"nome_completo":"Jennifer Widom","afiliacao":"Universidade de Stantord"},"Hector Garcia-Molina"]}

Validação de arquivos JSON

- Um documento JSON **bem formado** (ou **sintaticamente válido**) respeita os requisitos estruturais básicos:
  - Dados são representados por pares nome:valor; um valor pode ser um valor simples, um objeto ou um vetor; um objeto é formado por um conjunto de pares; um vetor é formado por uma lista de valores; etc.
- Um documento JSON **semanticamente válido** é bem formado e respeita um dado esquema
- Esquemas podem ser especificados por meio da linguagem

  JSON Schema
- Um documento contendo um esquema definido em JSON Schema é também um documento JSON válido
  - Semelhante ao que acontece com um esquema em *XML Schema*

JSON Schema – Exemplo (Arquivo “esquemaLivros.json”)

```
{ "$schema": "http://json-schema.org/schema#" }
{ "type":"object",
  "properties": {
     "Livros": {
        "type":"array",
        "items": {
           "type":"object",
           "properties": {
              "ISBN": { "type":"string", "pattern":"ISBN*" },
              "Preço": { "type":"integer", "minimum":0, "maximum":200 },
              "Edição": { "type":"integer", "optional": true },
              "Nota": { "type":"string", "optional": true },
              "Título": { "type":"string" },
              "Autores": {
                 "type":"array", "minItems":1, "maxItems":10,
                 "items": {
                    "type":"object",
                    "properties": {
                       "Nome": { "type":"string" },
                       "Sobrenome": { "type":"string" }}}}}}},
```

JSON Schema – Exemplo (Arquivo “esquemaLivros.json”)

```
    "Revistas": {
       "type":"array",
       "items": {
          "type":"object",
          "properties": {
             "Título": { "type":"string" },
             "Mês": { "type":"string",
                      "enum":["Janeiro","Fevereiro"] },
             "Ano": { "type":"integer" }}}}
}}
```

Linguagens de consulta para JSON

- Não existe linguagem de consulta padrão para JSON, mas já existiram várias propostas:
  - Jaql, JSON Path, SpahQL, JMESPath, ...
- Destaque: **JSONiq** – http://www.jsoniq.org/
  - Chamada de “SQL dos NoSQL”
  - Inspirada na XQuery
  - Possui os construtores “FLWOR”

JSONiq – Exemplos (1) Consulta (Junção)

Documento "users"

```
{
  "name" : "Sarah",
  "age" : 13,
  "gender" : "female",
  "friends" : [ "Jim", "Mary", "Jennifer"]
}
{
  "name" : "Jim",
  "age" : 13,
  "gender" : "male",
  "friends" : [ "Sarah" ]
}
```

Dois exemplos de consultas que retornam os objetos correspondentes aos amigos de Sara:

```
for $sarah in collection("users"),
    $friend in collection("users")
where $sarah.name eq "Sarah" and
      (some $name in $sarah.friends[]
       satisfies $friend.name eq $name)
return $friend
```

```
{
  "name" : "Jim",
  "age" : 13,
  "gender" : "male",
  "friends" : [ "Sarah" ]
}
```

```
let $sarah := collection("users")[$$.name eq "Sarah"]
for $friend in $sarah.friends[]
return collection("users")[$$.name eq $friend]
```

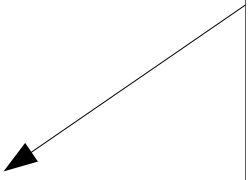
JSONiq – Exemplos (2) Transformação

Documento "satellites":

```
{
  "satellites" : {
    "AAU CUBESAT" : {
      "tle1" : "1 27846U 03031G 10322.04074654  .00000056  00000-0  45693-4 0  8768",
      "visible" : false
    },
    "AJISAI (EGS)" : {
      "tle1" : "1 16908U 86061A 10321.84797408 -.00000083  00000-0  10000-3 0  3696",
      "visible" : true
    },
    "AKARI (ASTRO-F)" : {
      "tle1" : "1 28939U 06005A 10321.96319841  .00000176  00000-0  48808-4 0  4294",
      "visible" : true
    }
  }
}
```

Consulta que sumaria os dados de satélites, mostrando quais estão visíveis e quais não estão:

```
let $sats := collection("satellites").satellites
return {
  "visible" : [
     for $sat in keys($sats)
     where $sats.$sat.visible
     return $sat
  ],
  "invisible" : [
     for $sat in keys($sats)
     where not $sats.$sat.visible
     return $sat
  ]
}
```



{ "visible" : ["AJISAI (EGS)", "AKARI (ASTRO-F)"], "invisible" : ["AAU CUBESAT"] }

Exemplos de SGBDs que usam o formato JSON

- MongoDB - https://www.mongodb.com/
  - BSON (Binary JSON)
- CouchBase - https://www.couchbase.com/
- CouchDB - https://db-engines.com/en/system/CouchDB
- MarkLogic Server - http://www.marklogic.com/
- Riak - https://riak.com/
- RethinkDB - https://rethinkdb.com/
- ...

Modelo Relacional X JSON

	Modelo Relacional	JSON
Estrutura	Tabelas	Conjuntos de vetores aninhados
Esquema	Fixo, predefinido	Autodescritivo, flexível
Consultas	Linguagem simples e muito expressiva (SQL)	Nenhuma linguagem padrão ou que seja amplamente usada
Implementação	Sistemas (SGBDs) nativos	Acoplada às linguagens de programação; Sistemas NoSQL

XML X JSON

	XML	JSON
Verbosidade	Mais	Menos
Complexidade	Mais	Menos
Validade	DTD, XML Schema - Muito usados	JSON Schema - Pouco usado
Linguagens de Consulta	XPath, XQuery, XSLT - “Padrões”, bastante usadas	JSON Path, Jaql, JSONiq, ... - Ainda muito “instáveis”

Referências bibliográficas

http://json.org/

- The JSON Data Interchange Format – ECMA Standard-404

  http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

- Material do curso de banco de dados da Universidade de Stanford

  http://www.class.stanford.edu/db/Winter2013/

- Validador de esquemas JSON online: http://jsonschemalint.com/