Document Databases

Introduction, JSON, XML

NoSQL Types

- Different types of noSQL databases
  - Key-value stores
  - **Document stores**
  - Column oriented (column family, wide-column) databases
  - Graph databases

Outline

- Text (document) data types
  - JSON (JavaScript Object Notation)
  - XML (eXtensible Markup Language)
- MongoDB as an example on Document Databases
  - data model
  - installation
  - operations
  - index structure

Data Format

- Structured Text Format
  - JSON, BSON (Binary JSON)
  - XML (eXtensible Markup Language)
  - RDF (Resource Description Framework)

JSON: Basics

- **J**ava**S**cript **O**bject **N**otations
  - derived/inspired by JavaScript scripting language
- Open standard for data interchange
  - support text and binary data
- Was designed with goal to be simple
  - both human and machine readable
- Universal
  - supported by many programming languages
    - you can read/write JSON files/objects with many programming languages
      - such as Java, Scala, Python, …

JSON: Basics

- File extension: *.json
- Uses objects and arrays data structures
- Internet media type (MIME type, or content type)
  - application/json
- JSON object is written as Text
  - so it can be easily sent from Browser to a server

http://www.json.org/

JSON: Examples

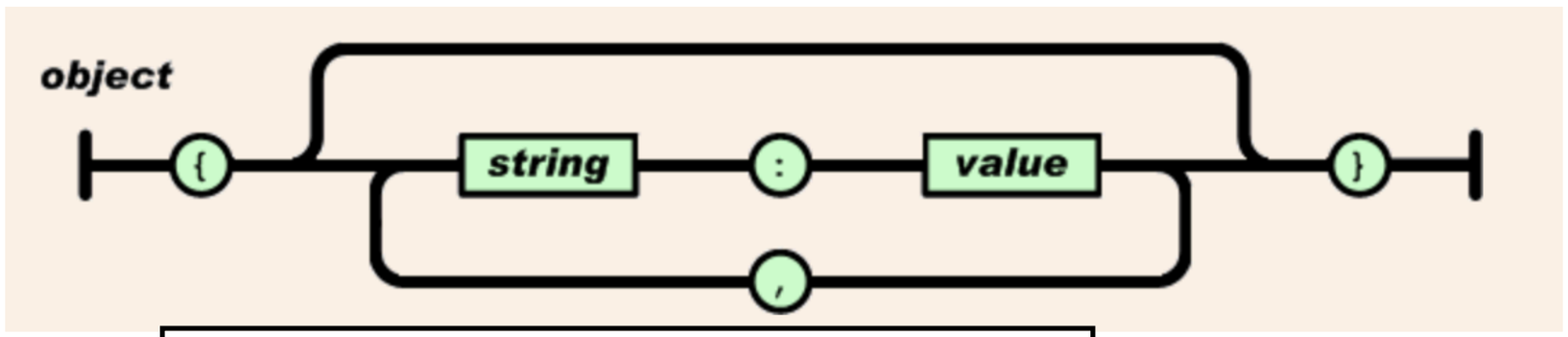
- Pay attention to opening and ending braces, square brackets
- Online tool for validating the script
  - for example https://jsonlint.com/

```
{
  "conferences":
  [
    {
      "name": "XML Prague 2015",
      "start": "2015-02-13",
      "end": "2015-02-15",
      "web": "http://xmlprague.cz/",
      "price": 120,
      "currency": "EUR",
      "topics": ["XML", "XSLT", "XQuery", "Big Data"],
      "venue": {
        "name": "VŠE Praha",
        "location": {
          "lat": 50.084291,
          "lon": 14.441185
        }
      }
    },

    {
      "name": "DATAKON 2014",
      "start": "2014-09-25",
      "end": "2014-09-29",
      "web": "http://www.datakon.cz/",
      "price": 290,
      "currency": "EUR",
      "topics": ["Big Data", "Linked Data", "Open Data"]
    }
  ]
}
```

JSON: Data structure

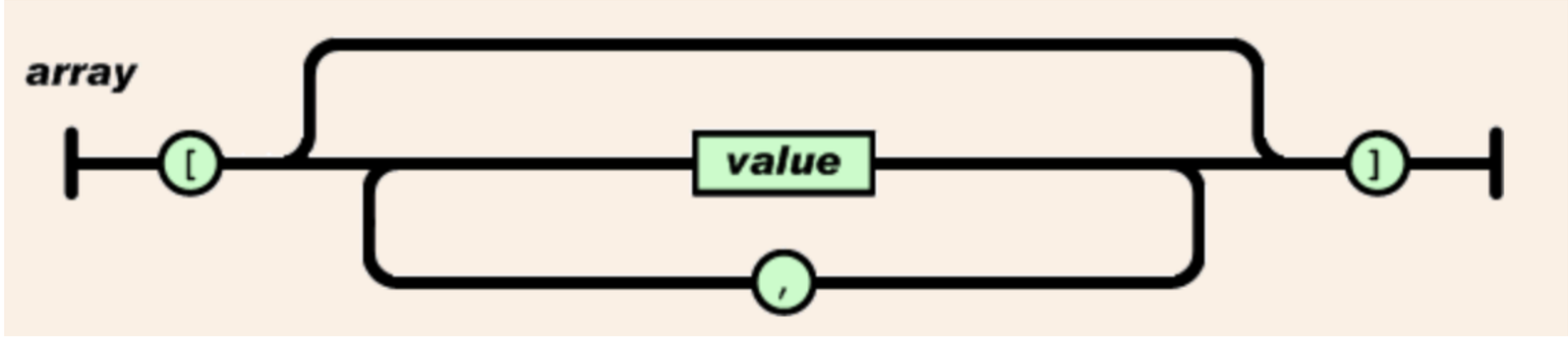
- JSON **object** is *unordered* collection of name-value pairs (properties)
  - begins with left brace { and ends with right brace }
  - name and value is separated with a colon :
  - and name-value pairs are separated with a coma ,




```
{ "name":"John", "age":30, "car":null }
```

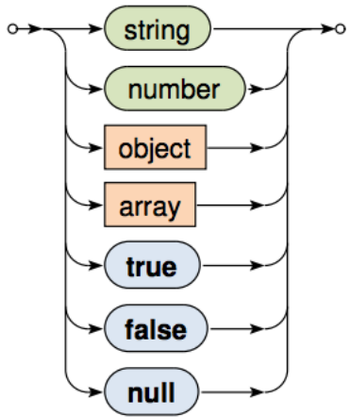
JSON: Data structure

- JSON **Array** *ordered* collection of values
  - correspond to structures such as arrays, list, sets, vectors, …
  - values can be of any type: string, number, boolean, array, object
  - duplicate values are allowed
  - Format: starts with opening left bracket and ends with right bracket, values are separated with comma



JSON: Data structure

- Value
  - String
    - enclosed between double quotation
  - Number: decimal integers, float
  - object
  - array
  - boolean



Access JSON : JS

- In Java script you can simply create the JSON object

```
{ "name":"John", "age":30, "car":null }
```

- to access one of the properties name/key
  - objectName.name

```
<script>
var myObj, x;
myObj =
{ "name":"John", "age":30, "car":null };
x = myObj.name;
</script>
```

Access JSON: Java

- You can create/access a JSON object using Java
- many third-party libraries
  - for example json-simple

```
<dependency>
            <groupId>com.googlecode.json-simple</groupId>
            <artifactId>json-simple</artifactId>
            <version>1.1.1</version>
        </dependency>
```

Java: create JSON example

- obj is defined as JSON object
  - {…}
  - contains name:values
    - added using put stmt
    - messages point to a JSON array

```
{
    "age":100,
    "name":"mkyong.com",
    "messages":["msg 1","msg 2","msg 3"]
}
```

Java: create JSON example

- obj is defined as JSON object
  - {…}
  - contains name:values
    - added using put stmt
    - messages point to a JSON array

```
{
    "age":100,
    "name":"mkyong.com",
    "messages":["msg 1","msg 2","msg 3"]
}
```

```
public class JsonSimpleWriteExample {

    public static void main(String[] args) {

        JSONObject obj = new JSONObject();

        obj.put("name", "mkyong.com");

        obj.put("age", new Integer(100));

        JSONArray list = new JSONArray();

        list.add("msg 1”);

        list.add("msg 2");

        list.add("msg 3");

        obj.put("messages", list);

        try (FileWriter file = new FileWriter("filename"))
{



            file.write(obj.toJSONString());

            file.flush();

        } catch (IOException e) {

            e.printStackTrace();

        }

    }

}
```

Java: Parse JSON example

- use the parser from simple-json library
- parser returns an Object, cast it to JSON Object
- you can access any name-value pair using the get(name)
- if the value is a JSON array, then you can iterate on its values

```
import org.json.simple.parser.ParseException;
public class JsonSimpleReadExample {
    public static void main(String[] args) {
     JSONParser parser = new JSONParser();
     try {
      Object obj = parser.parse(new
FileReader(“filename"));
      JSONObject jsonObject = (JSONObject) obj;
      String name = (String)
jsonObject.get("name");
      long age = (Long) jsonObject.get("age");
      // loop array
      JSONArray msg = (JSONArray)
jsonObject.get("messages");
      Iterator<String> iterator =
msg.iterator();
            while (iterator.hasNext()) {

System.out.println(iterator.next());
            }
        } catch (ParseException e) {
```

JSON Properties

- Does not allow adding comments
- Schema can be defined (JSON Schema)
  - specify the schema of the data
    - field names, data types
    - specify whether fields are required or optional
    - the schema is written in JSON format

XML: Basics

- eXtensible Markup Language
  - W3C standard, since 1996
  - both human & machine readable
  - structure data in between tags
    - opening / closing tags
    - user defined

```xml
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>


  <book category="cooking">

    <title lang="en">Everyday Italian</title>

    <author>Giada De Laurentiis</author>

    <year>2005</year>

    <price>30.00</price>

  </book>

</bookstore>
```

XML: Features

- Tags inside the XML file are user defined, conflict might happen
- Standard way to have a valid XML document
  - use XML schema
  - concept of name spaces
    - so usually tags names are preceded with prefix
      - the prefix is a URI
      - defined using **xmlns** (xml namespace) attribute

```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

XML: Features

- Technologies for parsing XML files
  - Document Object Model (DOM)
  - other technologies: XPath, XQuery
- Great usage for XML
  - configurations
  - meta-data
- Compared to JSON
  - JSON is more compact & easier to write

Parsing XML example

- define a parser
  - new DOMParser();
- has a function that convert XML text into DOM object
  - xmlDoc is DOM object
- elements can be found by tag name
  xmlDoc.getElementsByTagName

```
<script>
var parser, xmlDoc;
var text = "<bookstore><book>" +
"<title>Everyday Italian</title>" +
"<author>Giada De Laurentiis</author>" +
"<year>2005</year>" +
"</book></bookstore>";

parser = new DOMParser();                    mime type
xmlDoc = parser.parseFromString(text,"text/xml");

var title =
xmlDoc.getElementsByTagName(“title")[0].
childNodes[0].nodeValue;
</script>
```