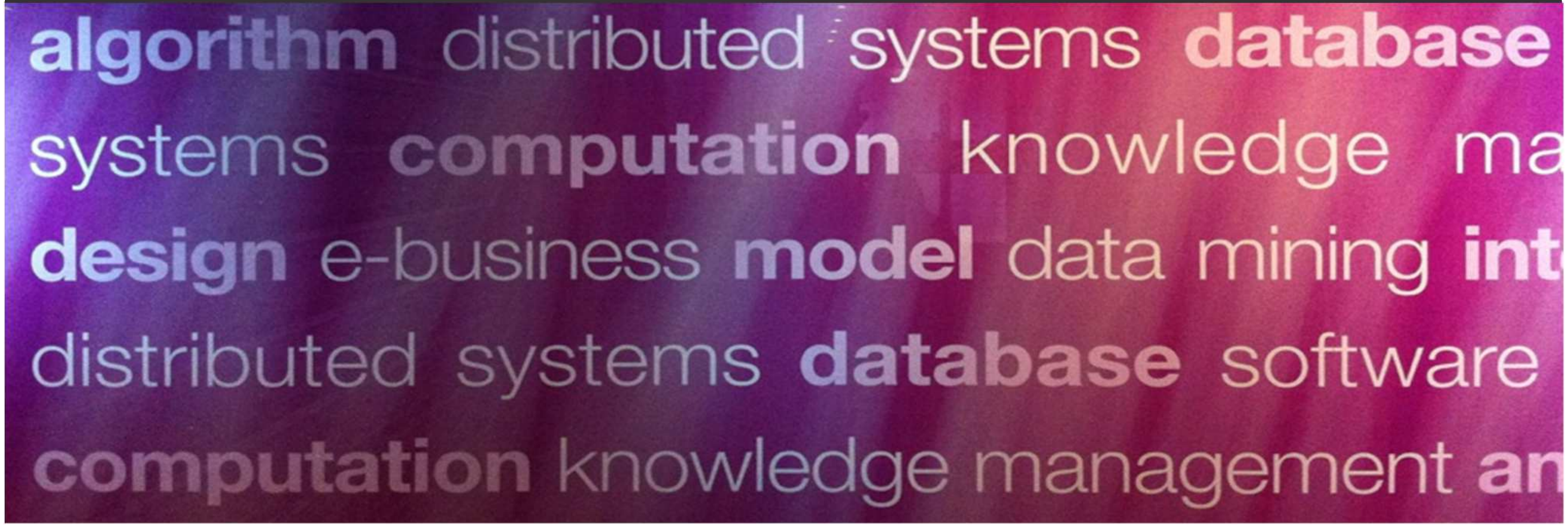
MONASH University

Information Technology

FIT3176 Advanced Database Design

Topic 11: JSON Documents and Schema

Dr. Minh Le
Minh.Le@monash.edu


algorithm distributed systems database
systems computation knowledge ma
design e-business model data mining int
distributed systems database software
computation knowledge management an

This unit…

1. Advanced Database Design
   - E/R is not complete enough
   - EER, covering superclass and subclass

2. SQL and PL/SQL
   - Trigger, Procedures/Functions, Packages

3. XML and XML DB
   - XML Documents
   - XML Schema
   - XML Database (XPath)

4. **JSON and JSON DB**
   - **JSON Documents and Schema** (this week)
   - JSON in Oracle (next week)

Learning Objectives

By the end of this week you should be able to:

- Describe what Big Data is and its key characteristics
- Understand and describe the structure of a JavaScript Object Notation (JSON) document
- Understand JSON syntax and data types
- Create a well-formed JSON document
- Construct a JSON schema
- Validate JSON documents using a JSON schema
- Understand the differences between XML and JSON data formats

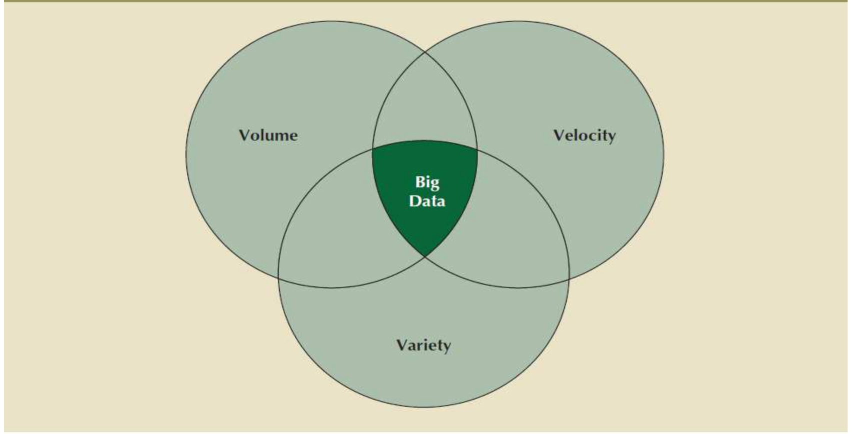
References

1. Smith, B. (2015). Beginning JSON. Apress.
2. Bassett, L. (2015). Introduction to JavaScript Object Notation A To-the-Point Guide to JSON. O'Reilly Media.
3. Sanjay Patni, Pro RESTful APIs - Design, Build and Integrate with REST, JSON, XML and JAX-RS, Apress, Berkeley, CA, 2017.
4. Coronel & Morris, Database Systems: Design, Implementation & Management, 2th Edition 2016 Chapter 14, Section 14-1.

Introduction to Big Data (1)

- Big Data is characterized by data of such volume, velocity, and/or variety that the relational model struggles to adapt to it.
- Big Data generally refers to a set of data that displays the characteristics of volume, velocity, and variety.
- Volume: A characteristic of Big Data that describes the quantity of data to be stored.
- Velocity: A characteristic of Big Data that describes the speed at which data enters the system and must be processed.
- Variety: A characteristic of Big Data that describes the variations in the structure of data to be stored.

FIGURE 14.2 CURRENT VIEW OF BIG DATA



Introduction to Big Data (2) - Volume

- Volume, the quantity of data to be stored, is a key characteristic of Big Data. The storage.
- To support the need for storing and processing ever-increasing data, there are two main methods to overcome this issue: scaling up and scaling out.
- Scaling up: A method for dealing with data growth that involves migrating the same structure to more powerful systems.
- Scaling out: A method for dealing with data growth that involves distributing data storage structures across a cluster of commodity servers.

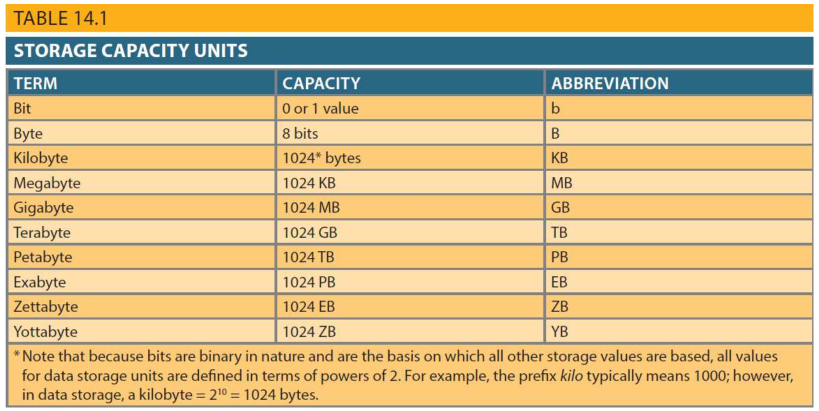


TABLE 14.1

STORAGE CAPACITY UNITS

TERM	CAPACITY	ABBREVIATION
Bit	0 or 1 value	b
Byte	8 bits	B
Kilobyte	1024* bytes	KB
Megabyte	1024 KB	MB
Gigabyte	1024 MB	GB
Terabyte	1024 GB	TB
Petabyte	1024 TB	PB
Exabyte	1024 PB	EB
Zettabyte	1024 EB	ZB
Yottabyte	1024 ZB	YB

* Note that because bits are binary in nature and are the basis on which all other storage values are based, all values for data storage units are defined in terms of powers of 2. For example, the prefix *kilo* typically means 1000; however, in data storage, a kilobyte = 2^{10} = 1024 bytes.

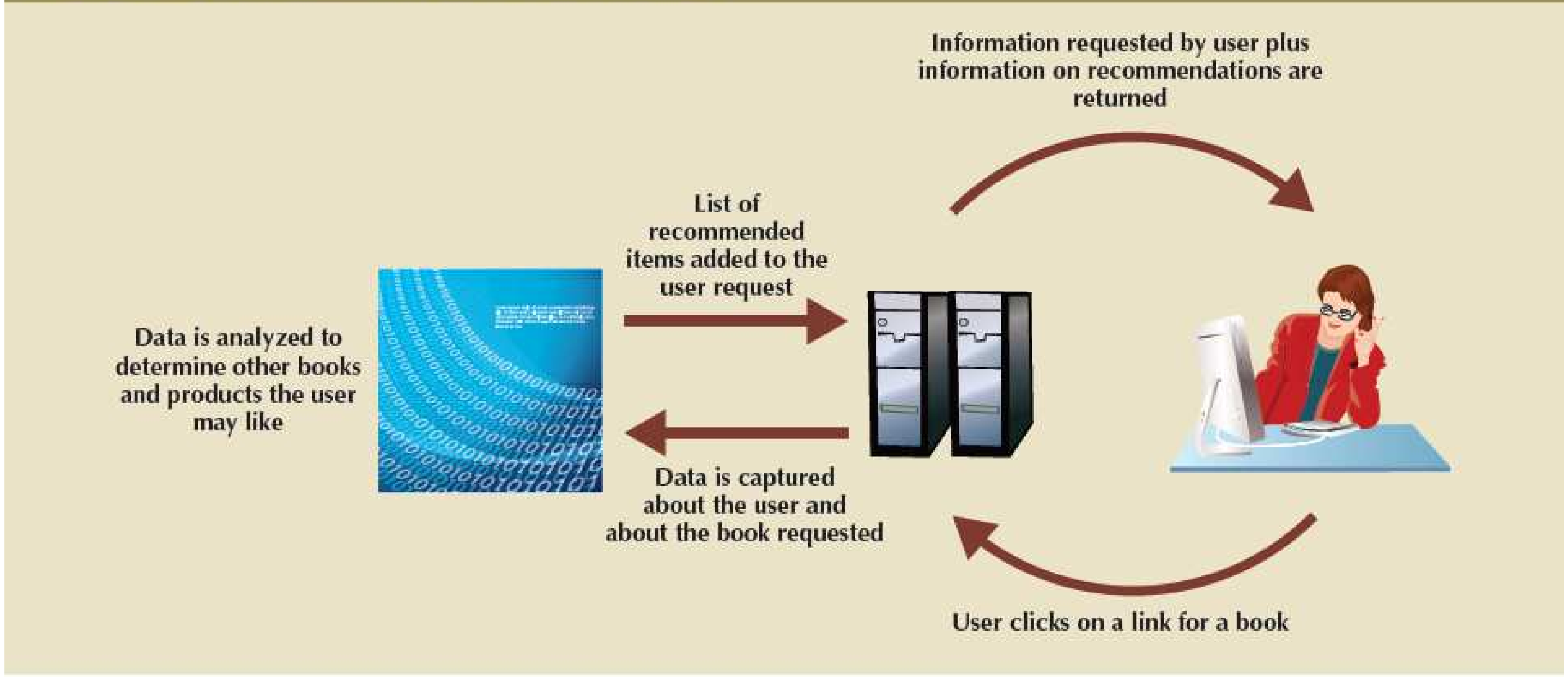
Introduction to Big Data (3) - Velocity

- Velocity, another key characteristic of Big Data, refers to the rate at which new data enters the system as well as the rate at which the data must be processed.
- The velocity of processing can be broken down into two categories: stream processing and feedback loop processing.
- Stream processing: the processing of data inputs in order to make decisions about which data to keep and which data to discard before storage.
- Feedback loop processing refers to the analysis of the data to produce actionable results. While stream processing could be thought of as focused on inputs, feedback loop processing can be thought of as focused on outputs.

Introduction to Big Data (4) - Velocity

Figure 14.3 shows a feedback loop for providing recommendations for book purchases.

FIGURE 14.3 FEEDBACK LOOP PROCESSING


Information requested by user plus information on recommendations are returned
List of recommended items added to the user request
Data is analyzed to determine other books and products the user may like
Data is captured about the user and about the book requested
User clicks on a link for a book

Introduction to Big Data (5) - Variety

- In a Big Data context, variety refers to the vast array of formats and structures in which the data may be captured. Data can be considered to be structured, unstructured, or semi-structured.
- Structured data: data that conforms to a predefined data model.
  – Relational databases rely on structured data. That is, a data model is created by the database designer based on the business rules.
- Unstructured data: data that does not conform to a predefined data model.
  – Unstructured data includes maps, satellite images, emails, texts, tweets, videos, transcripts, etc.
- Semi-structured data combines elements of both - some parts of the data fit a predefined model while other parts do not.
  – Data stored in XML and JSON data formats are called semi-structured data. For example, most web data are semi-structured data.

Introduction to JSON

- JSON stands for JavaScript Object Notation
- A lightweight data-interchange format
- The format originally specified by Douglas Crockford
- Derived from the ECMAScript Programming Language Standard
- A completely language independent text format
- Easy for humans to read and write
- Easy for machines to parse and generate
- The filename extension is .json.

*Source: www.json.org

JSON Applications

- JSON is increasingly becoming a standard data format for data driven applications.
- It is primarily used to transmit data between a server and web applications. E.g., Asynchronous JavaScript and JSON (or AJAJ) is a web development technique that provides the ability of a webpage to request new data after it has loaded into the web browser.
- Web services and APIs use JSON format to provide public data, e.g., Twitter has an API that can be queried to get Twitter data in JSON format.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is used when writing JavaScript based applications that includes browser extensions and websites.

*Source: https://en.wikipedia.org/wiki/JSON#Applications
https://www.tutorialspoint.com/json/json_overview.htm

Structures of JSON

- JSON, in a nutshell, is a textual representation defined by a small set of governing rules in which data is structured. JSON data can be organised in either of the following two structures
- A collection of string/value pairs. It is realised as an object, record, struct, dictionary, hash table, keyed list, or associative array.

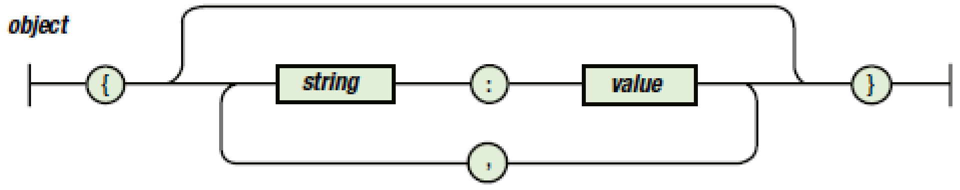


Figure 4-1. *Syntax diagram of a string/value pair collection*

- An ordered list of values. It is realised as an array, vector, list or sequence.

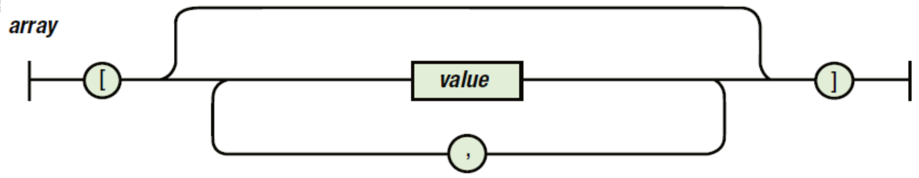
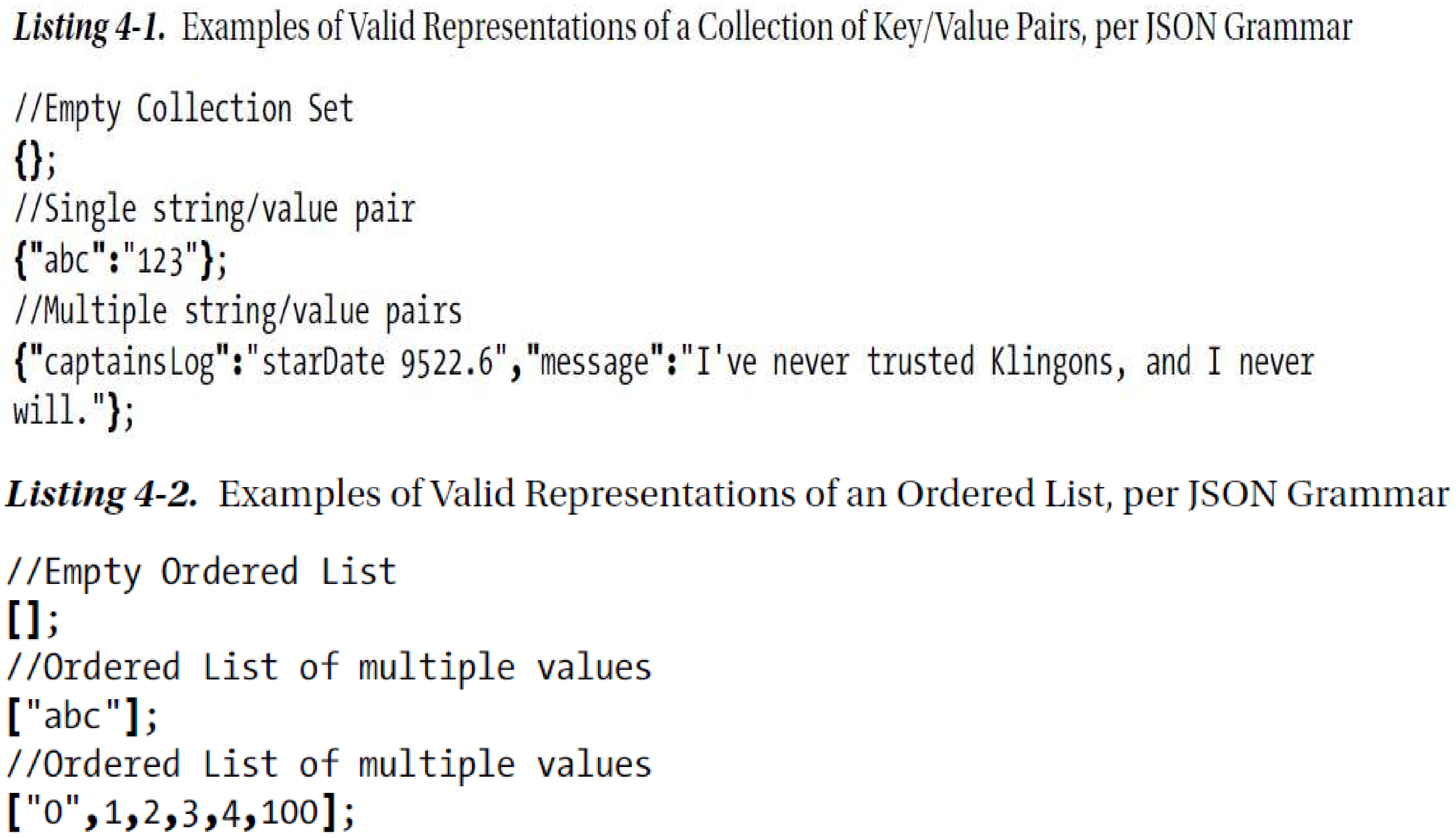


Figure 4-2. *Syntax diagram of an ordered list*

Examples of Valid Structures of JSON



Listing 4-1. Examples of Valid Representations of a Collection of Key/Value Pairs, per JSON Grammar

```
//Empty Collection Set
{};
//Single string/value pair
{"abc":"123"};
//Multiple string/value pairs
{"captainsLog":"starDate 9522.6","message":"I've never trusted Klingons, and I never
will."};
```

Listing 4-2. Examples of Valid Representations of an Ordered List, per JSON Grammar

```
//Empty Ordered List
[];
//Ordered List of multiple values
["abc"];
//Ordered List of multiple values
["0",1,2,3,4,100];
```

JSON Data Types

- Object
- String
- Number
- Boolean
- Null
- Array

JSON Data Types-Object

- JSON, at its root, is an object.
- JSON object is a list of name-value pairs surrounded in curly brackets ({ }).
- Pairs are separated by commas. There is a colon between the name and the value..

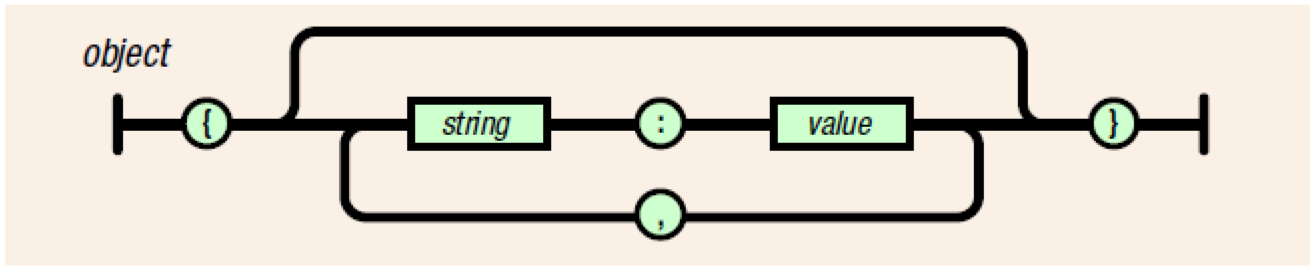


Figure 3-2. *JSON object*

*Source: [3]

Example of JSON Objects

- ➢ The top-level name-value pair is a person, with the value of an object.
- ➢ This person object has three name-value pairs: name, heightInCm, and head.
- ➢ The "name" property of the "person" object has a string value of "John Citizen".
- ➢ The "heightInCm" property of the "person" object has a number value.
- ➢ The "head" property of the "person" object has an object value.
- ➢ The "hair" property of the "head" object has an object value as well, with three string data typed name-value pairs : colour, length, and style. The "head" object also has an "eyes" property that has a string value of "brown".

```
{
    "person":{
        "name":"John Citizen",
        "heightInCm":175,
        "head":{
            "hair":{
                "colour":"black",
                "length":"short",
                "style":"curly"
            },
            "eyes":"brown"
        }
    }
}
```

JSON Data Types-String

- JSON string must be surrounded in double quotes.
- JSON string can include any Unicode characters.
- A valid string e.g., {"name":"Minh Le"}
- Backslash (\) character must be use to escape special characters
- E.g., single quote (\’), double quotes (\”), backlash (\\) forward slash (\/), backspace (\\b), form feed (\\f), tab (\\t), new line (\\n), carriage return (\\r), \\u followed by hexadecimal characters (e.g. the smiley emoticon \u263A).
- Valid JSON strings including special characters e.g.,

```
{
  "path" : "E:\\Data",
  "specialCharacters" : "New tab \\t, new line \\n, double quotes \")."
}
```

JSON Data Types-Number

- A number in JSON can be an integer, decimal, negative number, or an exponent
- A valid JSON object consisting of numbers

```
{
  "stockLevel" : 503,
  "totalCost" : 200.53,
  "seattleLatitude" : 47.606209,
  "seattleLongitude" : -122.332071,
  "earthsMass" : 5.97219e+24
}
```

JSON Data Types-Boolean

- Represents a logical value consisting of two possible values in all lowercase characters: true or false.
- A valid example of using Boolean data type

E.g.,

```
{
  "jsonIsGreat":true,
  "jsonRequireJavascript":false
}
```

JSON Data Types-Null

- Represents the intentional absence of a value. Null must be written in all lowercase characters.
- A valid example of using null data type
  E.g.,

```
{
   "studiedAtMonash":true,
   "age":28,
   "address":"1 Monash St, Clayton, VIC3168",
   "married":null
}
```

- The value of married cannot be specified, therefore it is null.

JSON Data Types-Array

- A collection or list of values, and the values can have a data type of string, number, Boolean, object or array.
- The values in an array are surrounded by square brackets ([ ]) and delimited by a comma.
- The syntax of JSON arrays is shown in Figure 3-3.

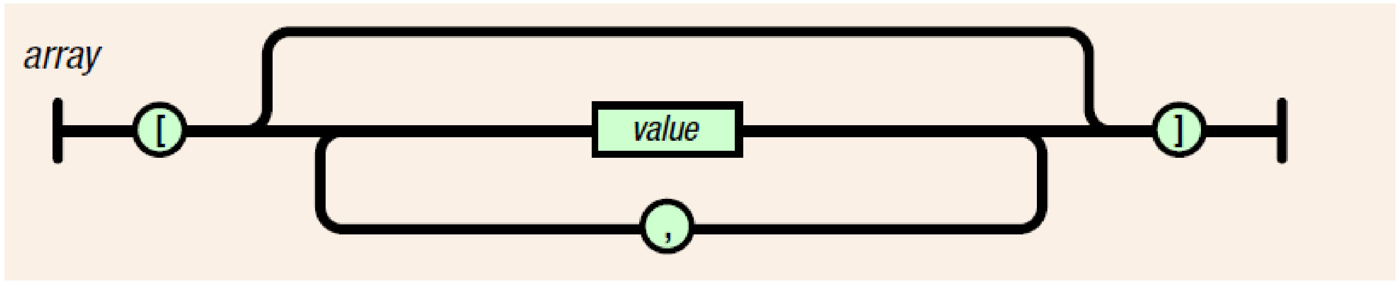


Figure 3-3. *JSON arrays*

*Source: [3]

Examples of JSON Arrays

- Valid examples of using Array data type. E.g.,

JSON staff array

```
{
  "staff":[
    "Jim Howards",
    "Bob James",
    "Tina Roberts",
    "Lynda
Williamson"
  ]
}
```

JSON ages array

```
{
  "ages":[
    30,
    35,
    25,
    20
  ]
}
```

Summary of JSON Data Format (1)

- The JSON array represents an ordered list of values, whereas the JSON object represents a collection of key/value pairs.
- Unordered collections of key/value pairs are contained within the following opening ({) and closing (}) curly brackets.
- The key of a member must be contained in double quotes.
- The key of a member and its associated value must be separated by the colon (:).
- Multiple values within an object or array must be separated by the comma (,) .
- Boolean values are represented using only lowercase true or false.

Summary of JSON Data Format (2)

- Number values are represented using double-precision floating number point format (e.g., 3 or 3.46). Number values can be specified with scientific notation.
- Control characters must be escaped via the backslash (\) (e.g., \n)
- Null values are represented as lowercase null.
- Arrays should be used when the key names are sequential integers.
- Objects should be used when the key names are arbitrary strings.
- JSON doesn’t specify date/ time data types. Date and time data should be stored as a string following a recommended standard format in ISO 8601 (e.g. date only: 2018-02-14, combined date and time in UTC: 2018-02-14T22:14:16+00:00).

JSON Schema

- A JSON schema is a virtual contract for data interchange.
- JSON schema is still under development. As of February 2018, JSON schema is still in draft-07.
- A JSON schema is written with JSON and can answer the following three questions for conformity validation:
    - Are the data types of the values correct? E.g., It is possible for us to specify that a value has to be a number, string, etc.
    - Does this include the required data? E.g., It is possible for us to specify what data is required, and what is not.
    - Are the values in the format that is required? E.g., It is possible for us to specify ranges, minimum and maximum of data values.

JSON Schema Syntax (1)

- An example of a JSON car object is shown below

```
{
   "manufacturer":"Porsche",
   "model":"911",
   "price":200000,
   “inStock":true
}
```

- A JSON schema begins with the following name-value pair.

```
{
   "$schema":"http://json-schema.org/draft-04/schema#"
}
```

JSON Schema Syntax (2)

An example of a JSON car object is shown below

```
{
  "manufacturer":"Porsche",
  "model":"911",
  "price":200000,
  "inStock":true
}
```

- The second name-value pair is the title. The third name-value pair is optional and is used to describe what this schema is for.

```
{
    "$schema":"http://json-schema.org/draft-04/schema#",
     "title":"Car"
    "description":"This schema is used to validate a JSON car object"
}
```

- The next name-value pair will define the properties that are included in the JSON.

```
{
   "$schema":"http://json-schema.org/draft-04/schema#",
   "title":"Car",
   "description":"This schema is used to validate a JSON car object",
   "properties":{
     "manufacturer":{
       "type":"string"
     }
}
```

- The completed JSON schema for the car object is shown in the next slide.

JSON Schema Syntax (3)

An example of a JSON car object is shown below

```
{
  "manufacturer":"Porsche",
  "model":"911",
  "price":200000,
  “inStock":true
}
```

```
{
  "$schema":"http://json-schema.org/draft-04/schema#",
  "title":"Car",
  "description":"This schema is used to validate a JSON car object",
  "properties":{
    "manufacturer":{
      "type":"string"
    },
    "model":{
      "type":"string"
    },
    "price":{
      "type":"number",
      "description":"Price of the car in US Dollars"
    },
    "inStock":{
      "type":"boolean",
      "description":"True means car is in stock"
    }
  }
}
```

JSON Schema - A Product Schema Example (1)

A Product using JSON object is described as in figure on the right

```
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

```
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object"
}
```

Step 1: A Product JSON schema is created as in figure on the left.

Source: http://json-schema.org/example1.html

JSON Schema - A Product Schema Example (2)

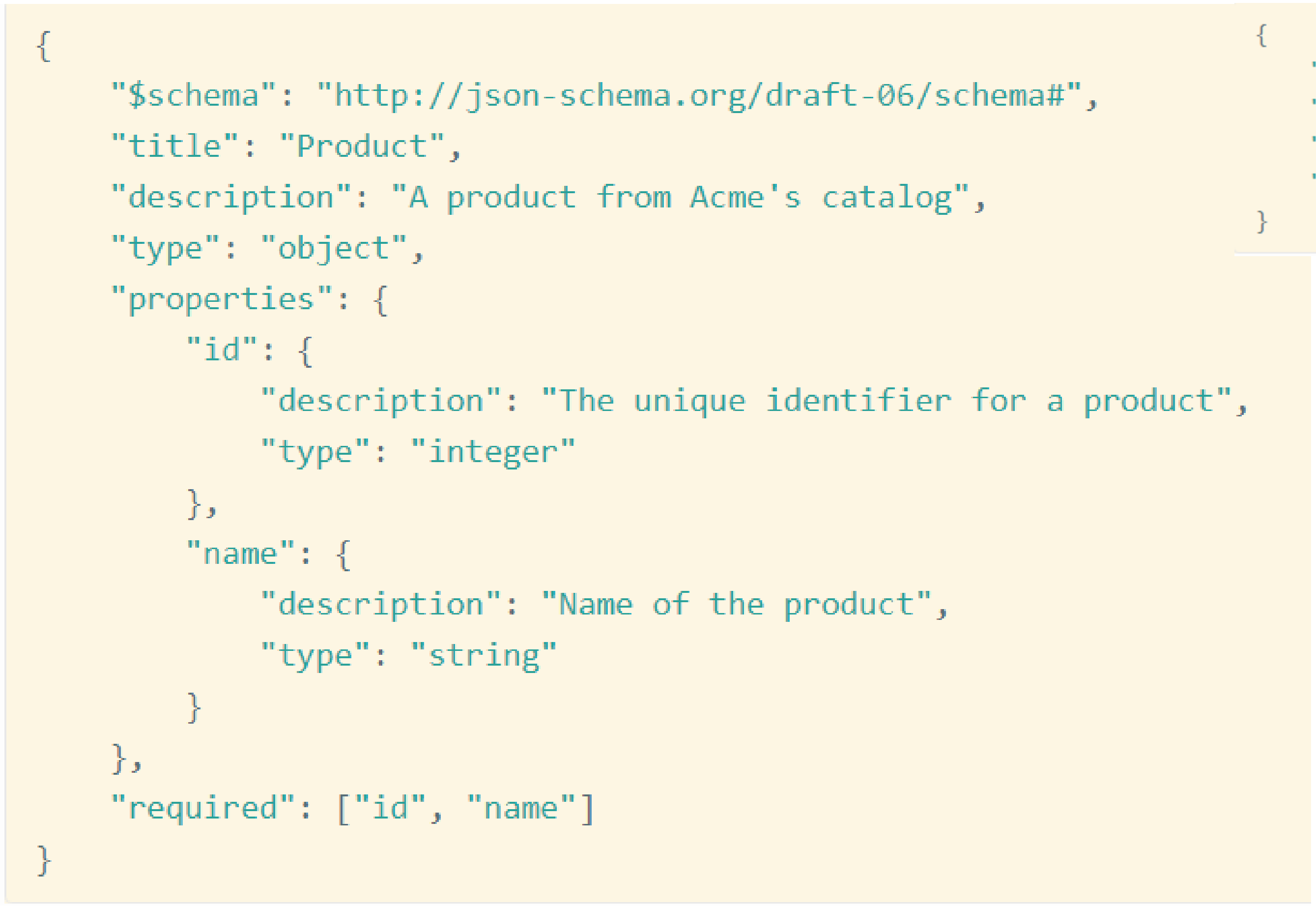
A Product using JSON object is described as in figure on the right

```
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

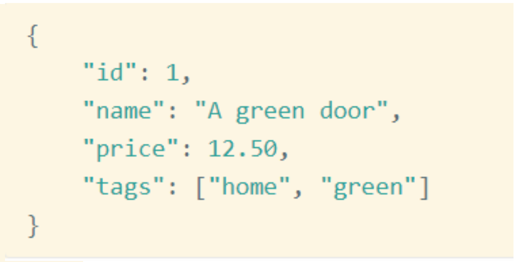
```
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object",
    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        }
    },
    "required": ["id"]
}
```

Source: http://json-schema.org/example1.html

JSON Schema - A Product Schema Example (3)



```
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object",
    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        },
        "name": {
            "description": "Name of the product",
            "type": "string"
        }
    },
    "required": ["id", "name"]
}
```



```
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

Source: http://json-schema.org/example1.html

JSON Schema - A Product Schema Example (4)

```
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object",
    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        },
        "name": {
            "description": "Name of the product",
            "type": "string"
        },
        "price": {
            "type": "number",
            "exclusiveMinimum": 0
        }
    },
    "required": ["id", "name", "price"]
}
```

```
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

Source: http://json-schema.org/example1.html

JSON Schema - A Product Schema Example (5)

```
{
    "id": 1,
    "name": "A green door",
    "price": 12.50,
    "tags": ["home", "green"]
}
```

```
{
    "$schema": "http://json-schema.org/draft-06/schema#",
    "title": "Product",
    "description": "A product from Acme's catalog",
    "type": "object",
    "properties": {
        "id": {
            "description": "The unique identifier for a product",
            "type": "integer"
        },
        "name": {
            "description": "Name of the product",
            "type": "string"
        },
        "price": {
            "type": "number",
            "exclusiveMinimum": 0
        },
        "tags": {
            "type": "array",
            "items": {
                "type": "string"
            },
            "minItems": 1,
            "uniqueItems": true
        }
    },
    "required": ["id", "name", "price"]
}
```

Source: http://json-schema.org/example1.html

Overview of JSON vs. XML (1)

- JSON data types are few and predefined. XML data can be either typeless or based on an XML schema or a document type definition (DTD).
- JSON has simple structure-defining and document-combining constructs: it lacks attributes, namespaces, inheritance and substitution.
- The order of the members of a JSON object literal is insignificant. In general, order matters within an XML document.
- JSON lacks an equivalent of XML text nodes (XPath node test text()). In particular, this means that there is no mixed content.
- JSON is most useful with simple, structured data. XML is useful for both structured and semi-structured data.

Overview of JSON vs. XML (2)

- JSON is generally data-centric, not document-centric; XML can be either.
- JSON is not a markup language; it is designed only for data representation. XML is both a document markup language and a data representation language.
- Because of its simple definition and features, JSON data is generally easier to generate, parse, and process than XML data.
- Use cases that involve combining different data sources generally lend themselves well to the use of XML, because it offers namespaces and other constructs facilitating modularity and inheritance.
- JSON, unlike XML (and unlike JavaScript), has no date data type. A date is represented in JSON using the available data types, such as string.