

CSCI235 Database Systems

JSON Schema

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Boolean](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

JSON Schema ? What is it ?

JSON Schema is a **vocabulary** for validation of JSON objects

JSON Schema describes the structures and types of existing JSON objects

JSON Schema allows for automated validation of JSON objects

JSON Schema itself is a JSON object

The latest version of JSON Schema is a version 2020-12

The following JSON object

```
{"city":"Sydney", "street":"Victoria", "building":25}
```

JSON object

validates well against the following JSON Schema

```
{ "type":"object",  
  "properties": { "city": { "type":"string" },  
                  "street": { "type":"string" },  
                  "building": { "type" : "number",  
                               "minimum": 1}  
                }  
}
```

JSON schema

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Boolean](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

Basics

The simplest possible **JSON schema** is the following (an empty object)

```
{ }
```

The simplest JSON schema

The schema given above validates every **JSON object**

Extensions of the schema given above impose the restrictions on the validated objects

For example, the following **JSON schema**

```
{ "type":"object",  
  "properties": { "Hello world message": { "type":"string" } }  
}
```

A Hello world JSON schema

validates well the objects

```
{ "Hello world message":"Hello world !" }
```

A Hello world object

```
{ "Hello world message":"Hello world !", "Greeting":"How are you ?" }
```

A How are you object

Basics

The following extension of the previous schema

An extended Hello world JSON schema

```
{ "type":"object",  
  "properties": { "Hello world message": { "type":"string",  
                                             "minLength": 1,  
                                             "maxLength": 13 }  
                }  
}
```

fails validation of an object

A Hello world object

```
{ "Hello world message":"Hello world !!!" }
```

because a string **Hello world !!!** is too long (15 characters)

It validates well the objects

A Hello world object

```
{ "Hello world message":"Hello world !" }
```

A How are you object

```
{ "Hello world message":"Hello world !", "Greeting":"How are you ?" }
```

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Boolean](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

Types

In **JSON schema** a **type** keyword is associated with a data type

JSON schema defines the following **basic types**: **string**, **Numeric types**, **object**, **array**, **boolean**, and **null**

A **basic type** associated with a keyword **type** can be either one of the types listed above or it can be an array of the types listed above

If **type** keyword is associated with a name of a **basic type** then a respective value must be of the same **basic type**, for example

```
{ "type": "string" }
```

Type string

It determines a type of a respective value as **string**

If **type** keyword is an **array** of **basic types** then each element of the array must be unique, for example

```
{ "type": [ "string", "number" ] }
```

Type string or number

It determines a type of a respective value as either **string** or **number**

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Boolean](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

String

A type `string` determines a type of a respective value as a string of characters

For example

```
{ "type": "string" }
```

String type

validates well any string of characters including an empty string

`string` type has the following restrictions: `minLength`, `maxLength`, `pattern`, and `format`

For example

```
{ "type": "string",  
  "minLength": 10,  
  "maxLength": 20 }
```

String type with length restrictions

validates well any string longer than 9 characters and shorter than 21 characters

String

A **pattern** restriction is used to match a string with a given regular expression

For example

```
{ "type": "string",  
  "pattern": "[1-9][0-9]" }
```

String type with regular expression restriction

validates well any string of characters representing a number in a range from "10" to "99"

A **format** restriction is used to match a string with predefined format

For example

```
{ "type": "string",  
  "format": "date" }
```

String type with format restriction

validates well any string of characters representing a date in a format "YYYY-MM-DD" for example "2020-07-21"

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Boolean](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

Numeric types

JSON Schema has two numeric types `integer` and `number`

A type `integer` is used for validation of integer numbers

For example

```
{ "type": "integer" }
```

Integer type

validates well any integer number like `-5`, `0`, `7`, etc

A type `number` is used for validation of any numeric value, either integer or floating point numbers

For example

```
{ "type": "number" }
```

Number type

validates well any integer or floating point number like like `-35.7`, `-7`, `0.0`, `23`, `23.4`, etc

Numeric types

Numeric types `integer` and `number` have the following restrictions:
`multipleOf`, `minimum`, `exclusive Minimum`, `maximum`,
`exclusiveMaximum`

For example

```
{ "type": "integer",  
  "multipleOf": 5 }
```

Integer type with multipleOf restriction

validates well any integer like `0`, `5`, `10`, `15`, etc

For example

```
{ "type": "number",  
  "multipleOf": 0.2 }
```

Number type with multipleOf restriction

validates well any number like `0.0`, `0.2`, `0.4`, `0.6`, etc

Numeric types

The restrictions `minimum`, `maximum`, `exclusiveMinimum`, and `exclusiveMaximum` can be used for expressing ranges

For example

```
{ "type": "integer",  
  "minimum": 0,  
  "exclusiveMaximum": 5 }
```

Integer type with a range restriction

validates well integer numbers 0, 1, 2, 3, and 4

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Boolean](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

Boolean

A type `boolean` determines a type of a respective value either as `true` or `false`

For example

```
{ "type": "boolean" }
```

Boolean type

validates well the values `true` and `false` and no other values

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Boolean](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

Null

A type `null` determines a type of a respective value as `null`

For example

```
{ "type": "null" }
```

Null type

validates well a lack of value `null` and no other values

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Boolean](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

Object

A type **object** determines a type of a respective value as **JSON object**

For example

```
{ "type":"object" }
```

Object type

validates well any object, like for example

```
{"city":"Sydney", "street":"Victoria", "building":25}
```

Flat JSON object

```
{ "name":"School of Astronomy",  
  "courses":[ {"code":"SOA101",  
                "title":"Astronomy for Kids",  
                "credits":3},  
               {"code":"SOA201",  
                "title":"Black Holes",  
                "credits":6}  
            ]  
}
```

Nested JSON object

```
{ }
```

Empty JSON object

Object

A key **properties** define the **key:value** pairs an object consists of

A key **properties** is associated with an object that consists of **key:value** pairs where each **key** is a name of a property and each **value** is a **JSON schema** used to validate a property

For example **JSON schema**

```
{ "type":"object",  
  "properties": { "city": { "type":"string" },  
                  "street": { "type":"string" },  
                  "building": { "type":"number",  
                               "minimum":1 }  
                }  
}
```

JSON schema

validates well the objects

```
{ "city":"Dapto","street":"Station","building":7}
```

JSON object

```
{ "city":"Dapto","street":"Station"}
```

JSON object

```
{ "city":"Dapto","street":"Station","building":7,"type":"skyscraper"}
```

JSON object

```
{ }
```

Empty JSON object

Object

A key `additionalProperties` determines whether additional properties not listed in **JSON schema** are allowed

For example **JSON schema**

```
{ "type": "object",  
  "properties": { "city": { "type": "string" },  
                  "street": { "type": "string" },  
                  "building": { "type": "number",  
                               "minimum": 1 }  
                },  
  "additionalProperties": false  
}
```

JSON schema

validates well the objects

```
{ "city": "Dapto", "street": "Station", "building": 7 }
```

JSON object

```
{ "city": "Dapto", "street": "Station" }
```

JSON object

```
{ }
```

Empty JSON object

Object

A key `requiredProperties` determines the compulsory properties of an object

For example **JSON schema**

```
{ "type": "object",  
  "properties": { "city": { "type": "string" },  
                  "street": { "type": "string" },  
                  "building": { "type": "number",  
                               "minimum": 1 }  
                },  
  "requiredProperties": ["city", "street"]  
}
```

JSON schema

validates well the objects

```
{ "city": "Dapto", "street": "Station", "building": 7 }
```

JSON object

```
{ "city": "Dapto", "street": "Station" }
```

JSON object

Object

The keys `minProperties` and `maxProperties` determine the minimum and maximum number of properties of an object

For example **JSON schema**

```
{ "type": "object",  
  "properties": { "city": { "type": "string" },  
                  "street": { "type": "string" },  
                  "building": { "type": "number",  
                               "minimum": 1 }  
                },  
  "minProperties": 2,  
  "maxProperties": 3  
}
```

JSON schema

validates well the objects

```
{ "city": "Dapto", "street": "Station", "building": 7 }
```

JSON object

```
{ "city": "Dapto", "street": "Station" }
```

JSON object

```
{ "street": "Station", "building": 7 }
```

JSON object

Object

A key **dependencies** determine the existence dependencies of one property on another

A value associated with a key **dependencies** is an object

Each entry in the object maps from a name of a property into an array of properties that are required whenever the property is present

For example **JSON schema**

```
{ "type": "object",  
  "properties": { "city": { "type": "string" },  
                  "street": { "type": "string" },  
                  "building": { "type": "number",  
                               "minimum": 1 }  
                },  
  "dependencies": { "street": ["building"] } }
```

JSON schema

validates well the objects

```
{ "city": "Dapto", "street": "Station", "building": 7 }
```

JSON object

```
{ "street": "Station", "building": 7 }
```

JSON object

```
{ "city": "Dapto", "building": 7 }
```

JSON object

Object

A key `patternProperties` determines a syntax of key names in an object

A value associated with a key `patternProperties` is an object

For example **JSON schema**

```
{ "type": "object",  
  "patternProperties": {  
    "^(city|CITY)": { "type": "string" },  
    "^(street|STREET)": { "type": "string" },  
    "^(building|BUILDING)": { "type": "number",  
                             "minimum": 1 }  
  },  
  "additionalProperties": false }  
}
```

JSON schema

validates well the objects

```
{ "city": "Dapto", "STREET": "Station", "building": 7 }
```

JSON object

```
{ "street": "Station", "BUILDING": 7 }
```

JSON object

```
{ "CITY": "Dapto", "BUILDING": 7 }
```

JSON object

```
{ }
```

JSON object

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

Array

A type `array` determines a type of a respective value as `array`

For example

```
{ "type": "array" }
```

Array type

validates well any array, like for example

```
[1, 2, 3, 4, 5, 6, 7]
```

Array of numbers

```
["ab", "cde", "efgh", "ijklm"]
```

Array of strings

```
[ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
```

Array of arrays

```
[ ]
```

Empty array

Array

A key `items` determines a type of values in an array

For example

```
{ "type": "array",  
  "items": { "type": "string" }  
}
```

Array type

validates well any array of strings like

```
["ab", "cde", "ab", "ijklm"]
```

Array of strings

```
["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"]
```

Array of strings

```
[ ]
```

Empty array

Array

Tuple validation determines a type of values when an array is a collection of items, each item has a different schema, and position of each item is important

For example

```
{ "type": "array",  
  "items": [ { "type": "string" },  
             { "type": "integer",  
               "minimum": 1 } ],  
  "additionalItems": false  
}
```

Array type with tuple validation

validates well any array of values like

```
["Station St.", 25]
```

A tuple

```
["Victoria St."]
```

Incomplete tuple

```
[ ]
```

Empty tuple

Array

A key `uniqueItems` requires each item in an array to be unique

For example

```
{ "type": "array",  
  "items": { "type": "string" },  
  "uniqueItems": true  
}
```

Array type

validates well any array of strings like

```
["ab", "cde", "efgh", "ijklm"]
```

Array of strings

```
[ "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" ]
```

Array of strings

```
[ ]
```

Empty array

Array

The keys `minItems` and `maxItems` determine a size of an array

For example

```
{ "type": "array",  
  "items": { "type": "string" },  
  "uniqueItems": true,  
  "minItems": 3,  
  "maxItems": 5  
}
```

Array type

validates well any array of strings like

```
["ab", "cde", "efgh", "ijklm"]
```

Array of strings

```
[ "0", "1", "2", "3", "4" ]
```

Array of strings

JSON Schema

Outline

[JSON Schema ? What is it ?](#)

[Basics](#)

[Types](#)

[String](#)

[Numeric types](#)

[Boolean](#)

[Null](#)

[Object](#)

[Array](#)

[Combined schemas](#)

Combined schemas

The keys `anyOf`, `allOf` and `oneOf` can be used to combine **JSON Schemas** together

For example `anyOf` keyword can be used to validate **JSON object** against one or more of the given schemas

```
{ "anyOf": [ {"type":"string",  
              "maxLength":5},  
              {"type":"boolean"} ]  
}
```

Validation with anyOf

validates well the objects like strings up to 5 characters long or Boolean values `true` and `false`

Combined schemas

For example **anyOf** keyword can be used to validate **JSON object** against one or more of the given schemas

```
{ "anyOf": [ {"type":"string",  
             "maxLength":5},  
             {"type":"string",  
             "minLength":3} ]  
}
```

Validation with anyOf

validates well any string

Combined schemas

The keys `anyOf`, `allOf` and `oneOf` can be used to combine **JSON Schemas** together

For example `allOf` keyword can be used to validate **JSON object** against all of the given schemas

```
{ "allOf": [ {"type":"string",  
             "maxLength":5},  
             {"type":"string",  
             "minLength":3} ]  
}
```

Validation with allOf

validates well any string no longer than 5 characters and no shorter than 3 characters

Combined schemas

The keys `anyOf`, `allOf` and `oneOf` can be used to combine **JSON Schemas** together

For example `oneOf` keyword can be used to validate **JSON object** against exactly one of the given schemas

```
{ "oneOf": [ {"type":"string",
              "maxLength":5,
              "minLength":5},
              {"type":"integer",
               "maximum":9,
               "minimum":1} ]
}
```

Validation with oneOf

validates well any string 5 characters long or any integer number in a range from 1 to 9 inclusive

References

[JSON Schema](#)

[Understanding JSON Schema](#)

[MongoDB - JSON Schema validation](#)

[MongoDB - \\$jsonSchema operator](#)