Course NDBI040: **Big Data Management and NoSQL Databases**

Lecture 06:

Data Formats

Martin Svoboda

10. 11. 2015

Charles University in Prague, Faculty of Mathematics and Physics

Outline

- Data formats
  - **JSON** – JavaScript Object Notation
  - **BSON** – Binary JSON
  - **XML** – Extensible Markup Language
  - **YAML** – YAML Ain't Markup Language
  - **RDF** – Resource Description Framework
  - **CSV** – Comma-Separated Values
  - **Protocol Buffers**
  - **Apache Thrift**

JSON

JavaScript Object Notation

Introduction

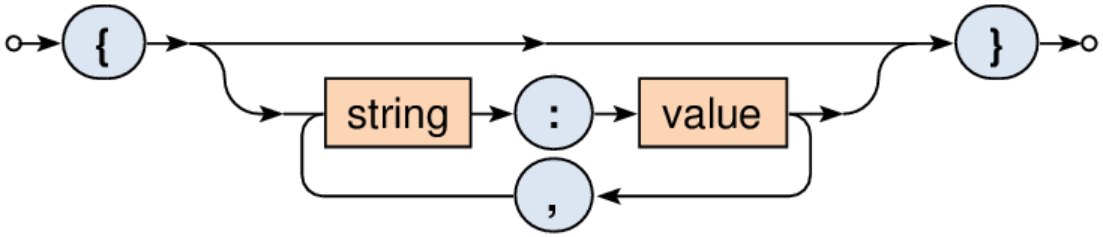
- **JSON** = JavaScript Object Notation
  - **Text-based easy-to-read-and-write open standard for data interchange**
    - Serializing and transmitting structured data
    - Design goals: **simplicity and universality**
  - Derived from JavaScript, but language independent
    - Uses conventions of the C-family of languages (C, C++, C#, Java, JavaScript, Perl, Python, …)
  - Filename: ***.json**
  - Media type: **application/json**
  - http://www.json.org/

Data Structures

- Built on two general structures
  - **Object**
    - Collection of name/value pairs
      - Realized as an object, record, struct, dictionary, hash table, keyed list, associative array, …
  - **Array**
    - List of values
      - Realized as an array, vector, list, sequence, …
  - All modern programming languages support them

Objects

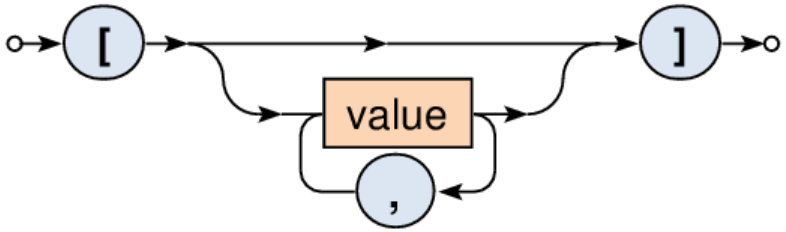
- **Object**
  - Unordered set of name/value pairs
    - Called properties of an object



  - Examples
    - `{"name" : "Peter", "age" : 30}`
    - `{}`

Arrays

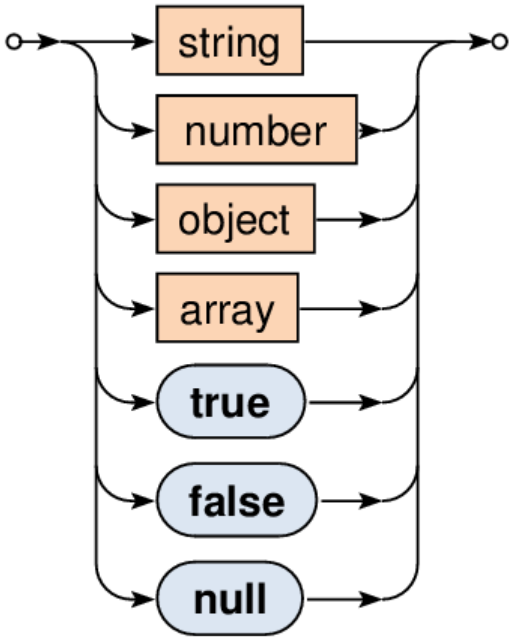
- **Array**
  - Ordered collection of values
    - Called items or elements of an array



  - Examples
    - `[3, 5, 7, 9]`
    - `[15, "word", -5.6]`
    - `[]`

Values

- **Strings**
- **Numbers**
- Nested **objects** or **arrays**
- **Boolean** values
  - `true` and `false`
- **Null** value
  - Missing information



Values

- **String**
  - Sequence of Unicode characters
    - Wrapped in double quotes
    - Backslash escaping sequences for special characters
  - Example: `"ab \n cd \" ef \\ gh"`
- **Number**
  - Integers or floating point numbers
    - Decimal system only
    - Scientific notation allowed
  - Examples: `10`, `-0.5`, `1.5e3`

Example

```
{
    "firstName" : "John",
    "lastName" : "Smith",
    "age" : 25,
    "address" : {
        "streetAddress" : "21 2nd Street",
        "city" : "New York",
        "state" : "NY",
        "postalCode" : 10021
    },
    "phoneNumbers" : [
        { "type" : "home", "number" : "212 555-1234" },
        { "type" : "fax", "number" : "646 555-4567" }
    ]
}
```

Schema

- **JSON Schema**
  - Defines the allowed structure of JSON data
    - For **validation** and/or **documentation** purposes
  - Design goals
    - Expressed in JSON
    - Self-describing
  - Available constructs
    - Structure of objects, arrays, their values, …
  - http://json-schema.org/

BSON

Binary JSON

Introduction

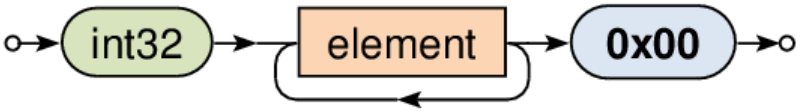
- **BSON**
  - **Binary-encoded serialization of JSON documents**
    - Allows embedding of JSON objects, arrays and standard simple data types together with a few new ones
  - MongoDB database
    - NoSQL database built on JSON documents
      - http://www.mongodb.com/
    - Primary data representation = BSON
      - Data storage and network transfer format
  - Filename: ***.bson**
  - http://bsonspec.org/

Example

- **JSON**
  - `{"hello" : "world"}`
- **BSON**
  - `\x16\x00\x00\x00` — Document size
    `\x02` — String data type
    `hello\x00` — Field name
    `\x06\x00\x00\x00world\x00` — Field value
    `\x00` — End of object

Grammar

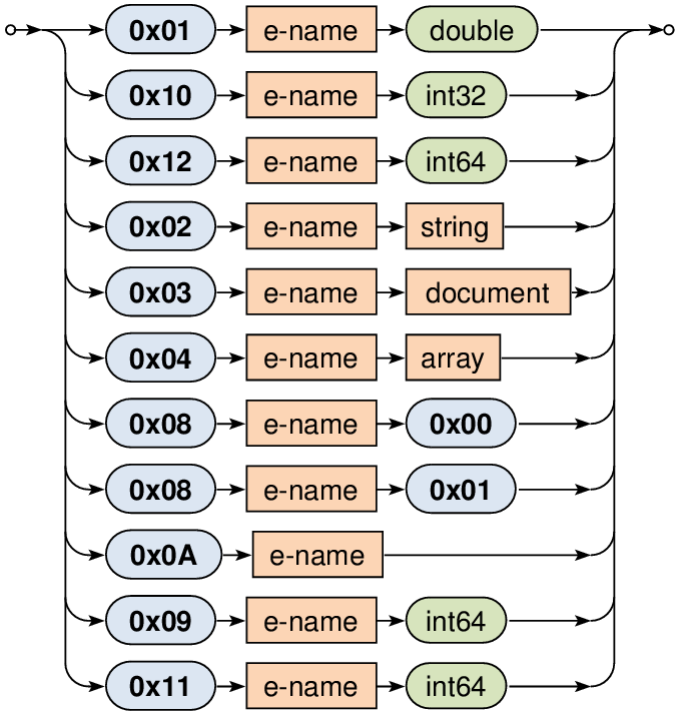
- **Document**
  - Encodes one JSON object / array / value
    - There can be more documents in one *.bson file
  - JSON array is first transformed into an object
    - E.g.: `["red", "blue"]` → `{"0": "red", "1": "blue"}`



  - Structure
    - **Total document size** in a number of bytes
    - Sequence of **elements**
    - Terminating 0x00

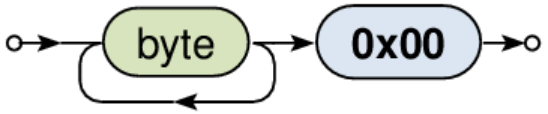
Grammar

- **Element**
  - Encodes one object property (name/value pair)
  - Structure
    - **Type** selector
      - 0x01 = double
      - 0x10 = 4B integer
      - 0x12 = 8B integer
      - 0x08 = boolean
      - 0x0A = null
      - 0x09 = datetime
      - 0x11 = timestamp
      - …
    - Field **name**
    - Field **value**

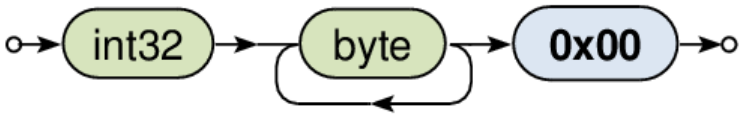


Grammar

- **Element name**
  - Unicode **string**
    - 0x00 not allowed inside
  - Terminating 0x00



- **String**
  - Total string **length**
  - Unicode **string**
  - Terminating 0x00



Grammar

- **Basic types**
  - `byte` – 1 byte (8-bits)
  - `int32` – 4 bytes (32-bit signed integer)
  - `int64` – 8 bytes (64-bit signed integer)
  - `double` – 8 bytes (64-bit IEEE 754 floating point)

XML

Extensible Markup Language

Introduction

- **XML**
  - **Extensible Markup Language**
    - Representation of semi-structured data
  - W3C recommendations
    - http://www.w3.org/TR/xml11/
  - Filename: ***.xml**
  - Media type: **text/xml**
  - Design goals
    - Simplicity, generality and usability across the Internet

Sample Document

```
<?xml version="1.0"?>
<library>
   <book id="1" catalogue="c1" language="en">
      <title>XPath</title>
      <author>John</author>
      <author>Peter</author>
   </book>
   <book id="2" catalogue="c1">
      <title>XQuery</title>
      <price>25</price>
   </book>
   <book id="3" catalogue="c2" language="en">
      <title>XSLT</title>
      <author>John</author>
   </book>
</library>
```

Structure

- Available constructs
  - **Elements**
  - **Attributes**
  - **Texts**
  - Comments, processing instructions, …
- Consistency issues
  - **Well-formedness**
  - **Validity** with respect to a schema

XML Schemas

- Schema languages
  - **Document Type Definition (DTD)**
  - **XML Schema (XSD)**
    - http://www.w3.org/TR/xmlschema-0/
  - **RelaxNG**
    - REgular LAnguage for XML Next Generation
    - http://www.relaxng.org/spec-20011203.html
  - **Schematron**
    - Assertions about the presence or absence of particular patterns in an XML document, typically based on XPath
    - http://www.schematron.com/

YAML

YAML Ain't Markup Language

Introduction

- **YAML**
  - Human friendly **data serialization standard** for all programming languages
    - **YAML Ain't Markup Language**
    - Originally: **Y**et **A**nother **M**arkup **L**anguage
  - Filename: ***.yaml**, ***.yml**
  - http://www.yaml.org/
  - Sample use-cases
    - Configuration files, debugging dumps, document headers, …

Sample Document

```
invoice: 34843
date: 2001-01-23
bill-to: &id001
    given: Chris
    family: Dumars
    address:
        lines: |
            458 Walkman Dr.
            Suite #292
        city: Royal Oak
        state: MI
        postal: 48046
ship-to: *id001
tax: 251.42
total: 4443.52
```

```
product:
    - sku         : BL394D
      quantity    : 4
      description : Basketball
      price       : 450.00
    - sku         : BL4438H
      quantity    : 1
      description : Super Hoop
      price       : 2392.00
comments:
    Late afternoon is best.
    Backup contact is Nancy
    Billsmer @ 338-4338.
```

Characteristics

- Design goals and features
  - **Common data structures**
    - Easily mapped to data types of agile high-level languages
      - Lists, associative arrays, scalars
    - Well-suited for **hierarchical data**,
      but also compact syntax for relational data
  - **Consistent model**
    - Ensures **portability** between programming languages
  - **Indented visual outline**
    - User-friendly serialization
    - **Easily readable** by humans

Characteristics

- Design goals and features
  - **Lean appearance**
  - **Missing enclosures**
    - Quotation marks, brackets, braces, open/close-tags
      - Because they can be hard for the human eye to balance
  - **Expressive and extensible**
  - **Easy to implement and use**
    - Supports one-pass processing

Basics

- **Unicode** encoding
- Basic primitives
  - **Mappings** – representing hashes, dictionaries, …
  - **Sequences** – representing arrays, lists, …
  - **Scalars** – strings, numbers, …
- Indentation-based scoping
  - No support for tabs
    - They must be replaced with spaces
- Content can be nested

Collections

- Collections
  - Use indentation for scope
  - Each entry begins on its own line
  - **Sequences**
    - Ordered collection of items
    - Each entry begins with a dash –
  - **Mappings**
    - Unordered collection of key/value pairs
    - Key and value parts are separated by a colon :

Collections: Examples

- **Sequence of scalars**

```
- Mark McGwire
- Sammy Sosa
- Ken Griffey
```

- **Mapping of scalars to scalars**

```
hr:  65    # Home runs
avg: 0.278 # Batting average
rbi: 147   # Runs batted in
```

Collections: Examples

- **Sequence of mappings**

```
-
  name: Mark McGwire
  hr:   65
  avg:  0.278
-
  name: Sammy Sosa
  hr:   63
  avg:  0.288
```

Collections: Examples

- **Mapping of scalars to sequences**

```
american:
  - Boston Red Sox
  - Detroit Tigers
  - New York Yankees
national:
  - New York Mets
  - Chicago Cubs
  - Atlanta Braves
```

Compact Collections

- Simplified syntax
  - **Flow style** instead of **block style** for collections
  - **Compact sequence**
    - Comma-separated list within **square brackets** `[ ]`
      - JSON arrays
    - Example
      - `[Mark McGwire, 65, 0.278]`
  - **Compact mapping**
    - Comma-separated list within **curly braces** `{ }`
      - JSON objects
    - Example
      - `{hr: 65, avg: 0.278}`

Compact Collections

- Simplified syntax
  - **Compact nested mapping**
    - Keys/value pairs can start immediately after `-`, `:` or `?`
    - Example

```
- item     : Super Hoop
  quantity: 1
- item     : Basketball
  quantity: 4
- item     : Big Shoes
  quantity: 1
```

Anchors and Aliases

- Representation of repeated nodes (objects)
  - First occurrences marked by **anchors** `&`
  - Additional occurrences referenced by **aliases** `*`
- Example

```
hr:
  - Mark McGwire
  - &SS Sammy Sosa
rbi:
  - *SS # Subsequent occurrence
  - Ken Griffey
```

Composite Keys

- **Complex keys** in mappings
  - Indicated using a question mark `?`
- Example: mapping between sequences

```
? - Detroit Tigers
  - Chicago cubs
:
  - 2001-07-23

? [ New York Yankees, Atlanta Braves ]
: [ 2001-07-02, 2001-08-12, 2001-08-14 ]
```

Logical Documents

- Represent **logical parts of data** in a file / stream
  - Directive `---` indicates start of a document
  - Directive `...` indicates end of a document
    - Without starting a new one, closing a stream connection etc.
  - Examples

```
---
- Mark McGwire
- Sammy Sosa
- Ken Griffey

---
- Chicago Cubs
- St Louis Cardinals
```

```
---
time: 20:03:20
player: Sammy Sosa
action: strike (miss)
...
---
time: 20:03:47
player: Sammy Sosa
...
```

Strings

- Flow styles
  - **Plain style**
    - Most examples so far
    - Example: `string`
  - **Double-quoted style**
    - For arbitrary strings
    - Standard escape sequences using a backslash `\`
    - Examples: `"string"` `"\x0d\x0a is \r\n"`
  - **Single-quoted style**
    - Only the quote can be escaped through `''`
    - Examples: `'string'` `' # Not a ''comment''.'`

Strings

- **Flow styles**
  - All flow scalars can span **multiple lines**, line breaks are always folded
  - Example

```
plain:
  This unquoted scalar
  spans many lines.

quoted: "So does this
  quoted scalar.\n"
```

Strings

- Block styles
  - **Literal style**
    - All line breaks are significant
      - i.e. they are preserved
    - **Indicated by |**
    - Example

```
# ASCII Art
|
  \//||\/||
  // ||  ||__
```

Strings

- Block styles
  - **Folded style**
    - Each line break is folded to a single space unless it ends an empty or a more-indented line
    - **Indicated by >**
    - Example

```
>
  Mark McGwire's
  year was crippled
  by a knee injury.
```

Strings

- Block styles
  - **Folded style**
    - Another example

```
>
  Sammy Sosa completed another
  fine season with great stats.

    63 Home Runs
    0.288 Batting Average

  What a year!
```

Data Types

- **Data types**
  - Basic
    - **Seq**, **map**, **str**, **int**, float, null, **binary**, omap, set, …
  - Application-specific
- Typing mechanism
  - **Tags**
    - Examples:
      - `number: !!int 123`
      - `string: !!str 123`
  - Untagged nodes are given a type depending on the application!

Data Types: Examples

- **Integers**
  - `canonical: 12345`
  - `hexadecimal: 0x0D`
- **Floating point numbers**
  - `canonical: 1.23015e+3`
  - `fixed: 1230.15`
  - `negative infinity: -.inf`
  - `not a number: .NaN`
- **Timestamps**
  - `date: 2002-12-14`
  - `canonical: 2001-12-15T02:59:43.1Z`
- **Miscellaneous**
  - `booleans: [ true, false ]`

Advanced Collections

- **Unordered set**
  - Represented as a mapping where each key is associated with a null value

```
!!set
? Mark McGwire
? Sammy Sosa
```

- **Ordered map**
  - Represented as a sequence of mappings, each describing right one key/value pair

```
!!omap
- Mark McGwire: 65
- Sammy Sosa: 63
```

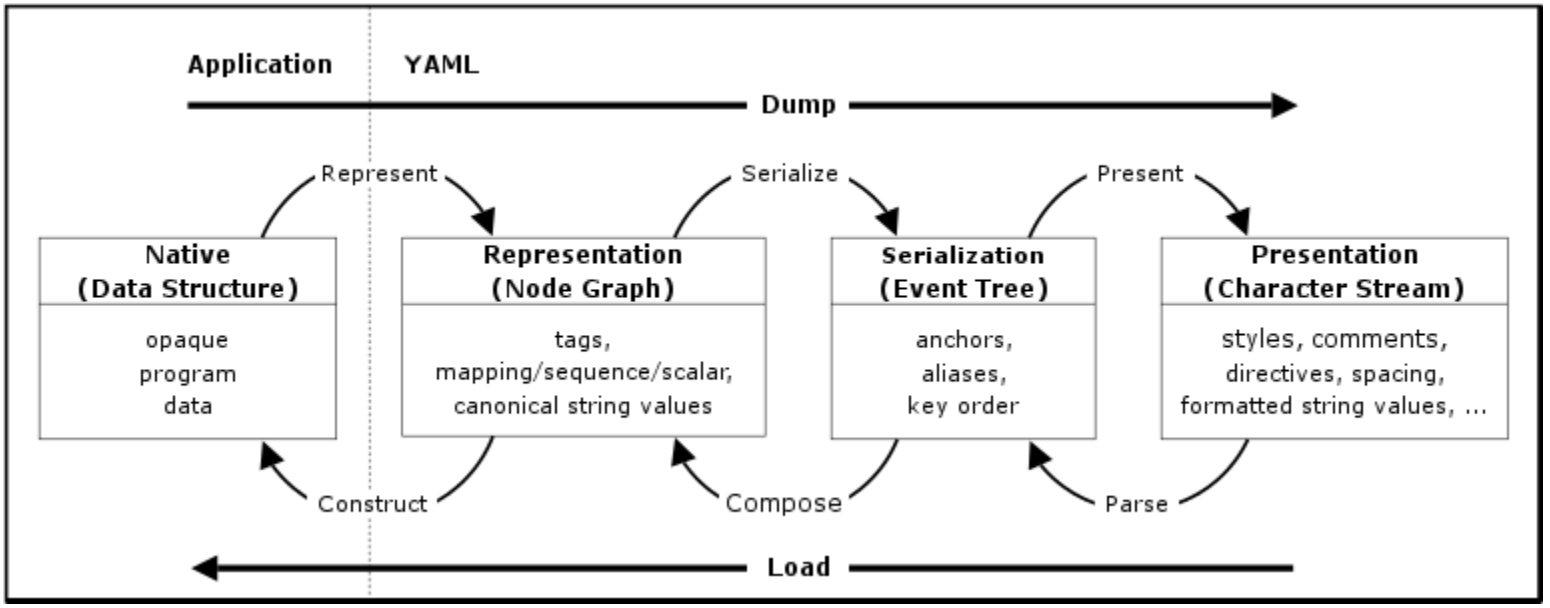
Bigger Example

```
invoice: 34843
date: 2001-01-23
bill-to: &id001
    given: Chris
    family: Dumars
    address:
        lines: |
            458 Walkman Dr.
            Suite #292
        city: Royal Oak
        state: MI
        postal: 48046
ship-to: *id001
tax: 251.42
total: 4443.52
```

```
product:
    - sku         : BL394D
      quantity    : 4
      description : Basketball
      price       : 450.00
    - sku         : BL4438H
      quantity    : 1
      description : Super Hoop
      price       : 2392.00
comments:
    Late afternoon is best.
    Backup contact is Nancy
    Billsmer @ 338-4338.
```

YAML Processor

- Required operations
  - **Dump** native data structures to a character stream
  - **Load** native data structures from a character stream



RDF

Resource Description Framework

Introduction

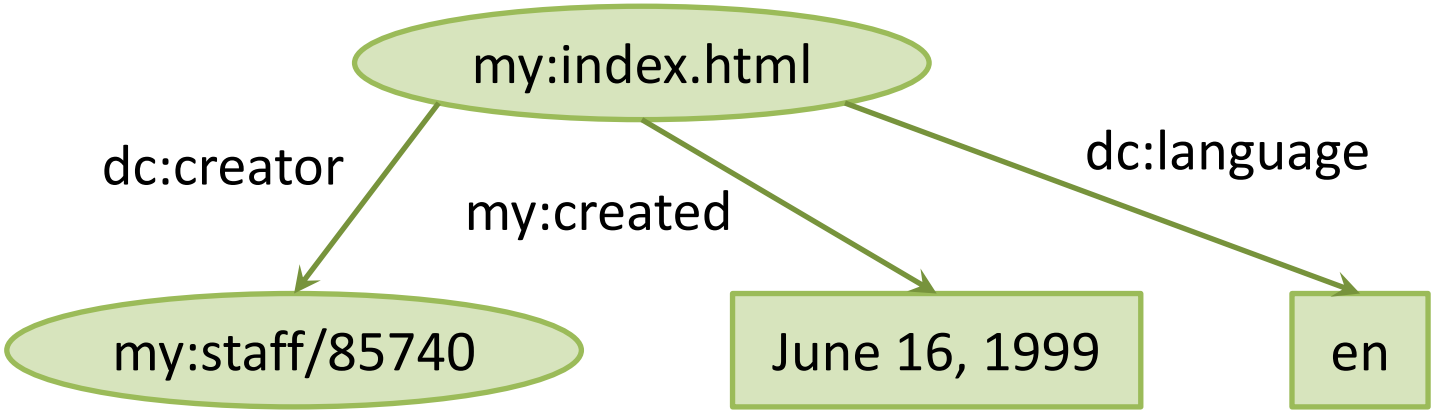
- **RDF**
  - **Resource Description Framework**
    - Language for representing information about resources in the World Wide Web
  - W3C recommendations
    - http://www.w3.org/TR/rdf-primer/
    - http://www.w3.org/TR/rdf-concepts/
    - http://www.w3.org/TR/rdf-syntax-grammar/
    - http://www.w3.org/TR/rdf-mt/

Statements

- Idea: statements about resources
  - **Resource**
    - Anything that is identifiable by a URI reference
      - Usually things identified by standard URLs…
      - … but also things that may not be directly retrievable
  - **Statement**
    - **Subject Predicate Object**
      - These triples are inspired by natural languages
    - Example
      - `http://is.cuni.cz/studium/sis#student358`
        `http://is.cuni.cz/studium/sis#name`
        `"John"`

Data Model: Example

```
my:index.html dc:creator my:staff/85740 .
my:index.html my:created "June 16, 1999" .
my:index.html dc:language "en" .
```



Data Model

- **Directed labeled multigraph**
  - **Vertices**
    - Represent subjects and objects
    - Labeled with their values
  - **Directed edges**
    - Represent particular statements (triples)
    - Labeled with values of predicates

Statements

- Components of RDF triples
  - **Subject**
    - Describes the thing the statement is about
    - *URI reference* or *blank node*
  - **Predicate**
    - Describes the property or characteristic of the subject
    - *URI reference*
  - **Object**
    - Describes the value of that property
    - *URI reference* or *blank node* or *literal*

Data Types

- **URI references**
  - URI with an optional fragment identifier
    - `http://is.cuni.cz/studium/sis#student358`
    - `mailto:svoboda@ksi.mff.cuni.cz`
    - `urn:issn:0167-6423`
  - Unicode characters
  - **Qualified names**
    - Similar idea as prefixes for namespaces in XML
      - `sis: = http://is.cuni.cz/studium/sis#`
      - `sis:student358 sis:name "John"`

Data Types

- **Blank nodes**
  - Special identifiers from the _ namespace
    - They are unique only within a given context
      - Particular document, database, …
- **Literals**
  - Plain or typed values
    - `"John"`
    - `"John"^^xsd:string`
  - Allowed only as objects

Serialization

- Existing approaches
  - **RDF notation**
  - **RDF/XML**
    - http://www.w3.org/TR/rdf-syntax-grammar/
  - **Notation3 (N3)**
    - http://www.w3.org/TeamSubmission/n3/
  - **Turtle notation**
  - **JSON-LD**
    - http://json-ld.org/
    - http://www.w3.org/TR/json-ld/

RDF Notation

- **Basic RDF Notation**
  - Syntax
    - Statements are terminated by dots .
    - Triple components are separated by spaces
    - URI references wrapped in angle brackets < >
    - Literal values in double quotes " "
  - Filename: ***.rdf**
- **Example**

```
<http://is.cuni.cz/studium/sis#student358>
<http://is.cuni.cz/studium/sis#name>
"John" .
```

Turtle Notation

- **Terse RDF Triple Language**
  - Subset of Notation 3
  - http://www.w3.org/TR/turtle/
  - Filename: ***.ttl**
  - Media type: **text/turtle**
- **Example**

```
@prefix my: <http://www.my.org/> .
my:s1 my:p1 my:o1 ,
              my:o2 ;
      my:p2 [ my:p3 my:o3 ] .
```

Vocabularies

- Schema languages
  - **RDF Schema (RDFS)**
    - Available constructs
      - Definition of **classes** and **properties**
      - Expected **domains** and **ranges** of properties
      - …
    - Not only **validation**, but also **inference of new information**
    - http://www.w3.org/TR/rdf-schema/
  - **Web Ontology Language (OWL)**
    - http://www.w3.org/TR/owl2-overview/

CSV

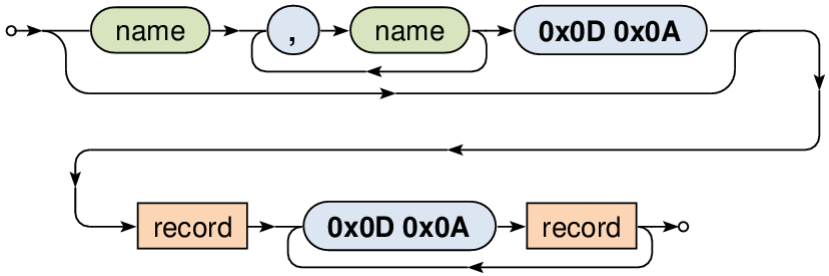
Comma-Separated Values

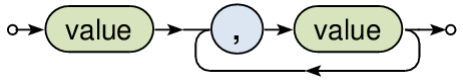
Introduction

- **CSV**
  - **Comma-Separated Values**
    - Not fully standardized
    - Different field separators (commas, semicolons)
    - Different escaping sequences
    - No encoding information
    - RFC 4180, RFC 7111
  - Filename: ***.csv**
  - Media type: **text/csv**

CSV Format

- Structure
  - **Document** = optional **header** + **records**
  - Record = comma separated list of **fields**





- Example
  - `Name,Age,City`
    `Peter,25,Prague`
    `John,30,Berlin`

Protocol Buffers

Introduction

- **Protocol Buffers**
  - Technique for serializing structured data
    - Developed by Google
    - Available under BSD license
  - Filename: ***.proto**
- Philosophy
  - **Describe** general structure of the data
    - Using an interface description language (ProtoBuf)
  - **Transform** this description into a source code in a particular programming language
    - Compilers for Java, C++, Python, JavaScript, PHP, …

Sample Document

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0; HOME = 1; WORK = 2;
  }
  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}

message AddressBook {
  repeated Person person = 1;
}
```

Apache Thrift

Introduction

- **Apache Thrift**
  - **Interface definition language** and **binary communication protocol**
    - Developed by Facebook
    - Available as open source (Apache)
  - Filename: ***.thrift**
- Philosophy
  - Analogous to Protocol Buffers
    - Compilers for C#, C++, Erlang, Haskell, Java, Perl, PHP, Python, Ruby, …

Sample Document

```
enum PhoneType {
  HOME,
  WORK,
  MOBILE,
  OTHER
}

struct Phone {
  1: i32 id,
  2: string number,
  3: PhoneType type
}
```