

1. přednáška – ÚVOD

- Konceptuální model
 - Třída reprezentuje typ – zvíře, adresa
 - Atributy třídy – instance dané třídy má nějaké vlastnosti
 - Asociace/vazby – mají název a multiplicitu
- Datový model
 - Je to logické zobrazení dat (sada nástrojů pro reprezentaci dat v dané doméně)
 - RDF je grafový datový model -> zobrazuje data v grafu
 - Další grafový datový model je LPG (labeled property graph)
 - Dále to mohou být hierarchie/stromy
 - DOM – document object model
 - Datový model JSON (je to formát i model) -> také stromový
 - Dále to mohou být (relační) tabulky
- Datový formát
 - Je to fyzický pohled na nějaká data (serializovaná data do souborů), váže se již s konkrétním datovým modelem
 - Vyberu si RDF grafový model
 - Lze ho serializovat do **textových formátů**: N-Triples, N-Quads, Turtle
 - Vyberu si LPG
 - Daná data pak mohou serializovat do CSV, JSONu
 - Vyberu si hierarchický datový model (např. DOM nebo JSON)
 - DOM lze serializovat do XML, HTML
 - JSON lze serializovat do JSON, XML
 - Vyberu si relační datový model
 - Lze ho reprezentovat do CSV, SQL dump
- Datové schéma
 - Jsou to anotace a omezení aplikovatelná na instance dat, která data popisují a mohou si pomocí nich také data validovat
 - CSV má CSV on the web
 - RDF má SHACL (nebrali jsme)
 - JSON má JSON Schema
 - XML má XML Schema (pak i Relax NG)
 - Metaformát = hostitelský formát
 - Datová schémata se právě používají pro definici specifických formátů v rámci těch hostitelských
 - Příklady
 - GeoJSON -> prostorové informace v rámci JSONu
 - GTFS -> reprezentace jízdních řádů v rámci CSV
 - SVG -> vektorová grafika v rámci XML
- Obecné vlastnosti datových formátů
 - Otevřené a uzavřené formáty
 - Otevřené – specifikace formátů je na internetu a dostupná všem

NPRG036 – Datové formáty

- Metaformáty – xml, json, csv, rdf
 - Specifické formáty – geojson, svg
 - Uzavřené – specifikace není dostupná, je třeba za to platit a je třeba se registrovat
 - Např. railML, je to xml based a slouží pro popis železnic
- Machine-readable format
 - Není to vlastnost formátu, ale spíše konkrétní způsob konkrétního použití daného formátu
 - Říká, jestli jsou data jednoduše zpracovatelná aplikací pro práci s daným formátem
- Binary vs text-based formats
 - Binární soubor – struktura je popsána bit po bitu
 - Nejsou čitelné textovými editory
 - Mohu použít gex editor
 - Textový soubor – obsahuje text, strukturované jako znaky na řádcích
 - Čitelné textovými editory
 - Textový soubor ale pro bity používá kódování znaků
 - US-ASCII – znak je 7 bitů
 - UTF-8 – používá se ve všech rozumných datových formátech, jeden znak je 1 až 4 bajty, emotikony používají 4 bajty
 - BOM – byte order mark, má 3 bajty, pokud je toto v souboru, tak to znamená, že je použito nějaké konkrétní kódování
 - Většina formátů používá UTF-8 bez BOM -> protože se používá hlavně obecně UTF-8, tedy se na to lze poměrně spolehnout
- Standardy
 - IETF – vymýšlí standardy, internet engineering task force
 - ISOC – zabývá se spíše politickým vedením internetu, internet society
 - W3C – mezinárodní standardy pro www, world wide web consortium
 - Placené členství, dělají doporučení pro html, css atd.
 - ICANN – spravuje IANA, hodně zaměřené na internet a komunikaci, typy medií
 - Další jsou – ECMA International
 - RFC 2119 – je to specifikace toho, jak psát specifikace a jaká klíčová slova v nich používat (RFC = request for comments)
- Identifikátory
 - URI – Uniform resource identifier
 - Je to jen id, nemá fci (nemusí fungovat v browseru)
 - URN – ... name
 - Typ uri
 - Nemá nic společného s lokalizací daného resourcu
 - URL – ... locator
 - Vždy funguje v browseru a najde ho
 - IRI – Internationalized Resource Identifier
 - Dovolí použít utf-8, URI ani URL ne
 - URI jsou tedy zakódované procenty (mezera = %20)

- Datové typy
 - o Jsou stejné napříč datovými formáty
 - o Booleans, numbers (ints, decimals, float, double, date, time, dateTime, timeZones)

2. přednáška – RDF, RDFS, Linked Data

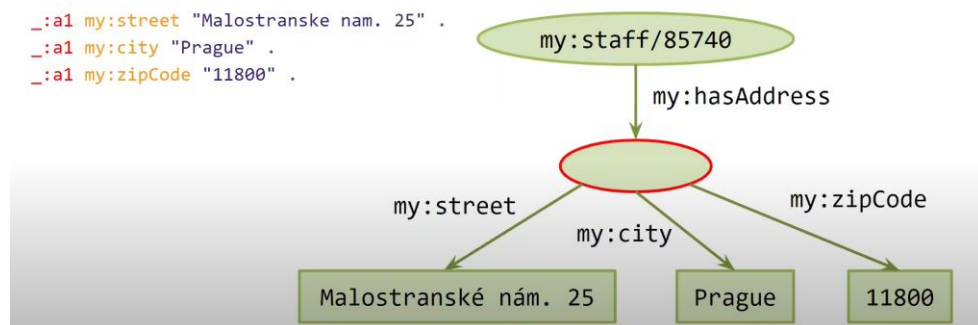
- V první části se mluví obecně o grafové reprezentaci dat (do 11:50)
- RDF = grafový datový model, orientovaný pojemnovaný multigraf
 - o Je to set trojic – subjekt, predikát/vlastnost, objekt
 - o Objekt je buď něco identifikovatelné IRI nebo literál
 - o Literál má ale 2 části – textová hodnota („Honda“) a datový typ (xsd:Date)
 - Jsou tam i literály s language tagem (vlastně druhý typ literálu)
 - o Blank nodes – prázdný uzel bez IRI

```
my:staff/85740 my:hasAddress _:a1 .
```

```
_:a1 my:street "Malostranske nam. 25" .
```

```
_:a1 my:city "Prague" .
```

```
_:a1 my:zipCode "11800" .
```



- o
- o Serializace RDF
 - N-Triples – trojice, kde na pozici objektu může být IRI, literál s langTagem, nebo s datovým typem
 - Velmi jednoduchá, lze lehce číst, zpracovávat
 - Je to velmi zdlouhavé, zbytečné info (samé IRIs)
 - Mnoho problémů N-Triples řeší RDF Turtle
 - Prefixy – sdružuje IRI, redukuje pak jejich psaní
 - ; a . – středník říká, že další trojice je o stejném subjektu, tečka ukončuje
 - Čárka zase umožňuje výčet pro stejné subjekt a predikát (více roků opravy auta, jako jsme měli)
 - Relativní IRI
 - Máme @base, pomůže mi zadefinovat začáteční adresu, podle které pak mohu stavět relativní
 - Rdf:type lze zkrátit na a -> ex:White a ex:Color
 - Jak řešit to, kdybych chtěl něco říct o celé trojici?
 - Pojmenované grafy – z RDF triples se stávají RDF quads

- Čtvrtá část je právě graf, tedy čtvrtým výrazem popisují nějakou množinu (graf) dalších trojic
- RDF dataset – skládá se z množiny pojmenovaných grafů a jednoho výchozího grafu
- RDF TriG – rozšíření serializace Turtle o podporu pojmenovaných grafů
- RDF schema – slouží pro zadefinování vlastních slovníků, typů
 - Mluví o typech, jako jsme dělali v úkolu (raději se podívat)
 - Rdf properties mohou existovat nezávisle na třídách
- RDF má i položku rdf:List, funguje jako spoják (first a rest)
 - V Turtle má zkratku ()
- OWA – open world assumption
 - Je to princip, kdy pokud nemám nějaká data, tak řeknu, že nelze určit (narozdíl od SQL, kde pokud v tabulce nemám, že Paul je muž, tak řeknu, že Paul není muž)
- RDF se používá na linked data (propojená data)
 - Problémy s nepropojenými daty:

Issues with regular, non-linked data

i.e. CSV, JSON, XML, Excel files...

ID	Name	Stars
1235	Joaquin Phoenix	Joker
1234	Robert De Niro	Joker
...		

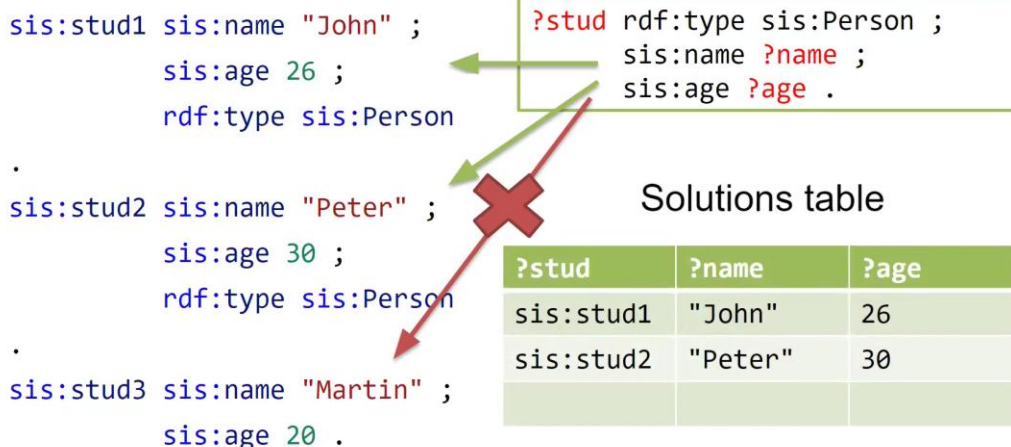
- Ambiguous identification of entities in data
 - Person with ID aaa1234 in a document located on my laptop in folder /data/temp/people.json
 - Another person with ID aaa1234 in the XML file on this CD
- Low findability and accessibility of data
 - Get data about person aaa1234 => Go to my laptop, open the folder, load/open the file, search/query
- No contextual information
 - Person aaa1234 lives in Prague. I want to know more about Prague. Where and how do I get the information?

- Cíle linked data
 - Udělat machine readable a srozumitelná data
 - Přidávat k datům i jejich kontext, tedy propojovat je is s jinými
- Principy linked data
 - Vše budeme identifikovat pomocí URI
 - Nejlépe vše musí být http URI
 - URI musí nabízet standardy (rdf, sparql)
 - Přidat linky i k dalším relevantním URIs

3. přednáška – SPARQL

- Na začátku byl yapping o linked data
- SPARQL – dotazovací jazyk nad daty v datovém formátu RDF
- Sparql endpoint – http webová služba, posílají se tam sparql queries
- Proměnná – začíná otazníkem a pak jméno variable -> ?name

SPARQL querying - more triples



- Pokud bych chtěl vrátit Martina, i když tam nemá, že je Person, tak mohu použít OPTIONAL -> OPTIONAL { ?stud rdf:type ?type }
- UNION – množinové sjednocení, musí matchnout alespoň 1 část
- Syntaxe SPARQL je velmi podobná RDF turtle, tedy na začátku také musí být známé prefixy pro ta data
- Funguje tam i base -> je to ale sračka, nepoužívat
- Blank nodes – pokud je to blank node v datech, tak ve výsledku dotazu bude mít random id
- GRAPH – slouží pro dotazování nad n-quads
- FILTER – píše se tam omezení pro řádky (?age > 27)
- LIMIT – omezení počtu vrácených řádků ve výsledku
- BIND – lze tím spočítat hodnotu, která není v databázi a vrátit jí (cena + sleva)
- Funkce na RDF termech

SPARQL - functions on RDF terms

- isIRI
- isBlank
- isLiteral
- isNumeric
- str
- lang
- datatype
- IRI
- STRDT
- STRLANG
- UUID

Examples of function usage:

```
UUID() <urn:uuid:b9302fb5-642e-4d3b-af19-29a8f6d894c9>
```

```
STRLANG("chat", "en") "chat"@en
```

- Pak jsou tam funkce i pro stringy, number, dates, times
- Jako v SQL můžeme vnořovat queries do queries
- ASK – kontroluje, jestli existuje alespoň jeden výsledek pro daný dotaz, výsledek je true nebo false, ne tabulka jako u select

- DESCRIBE – vrací RDF graf o daném subjektu nebo objektu
- ORDER BY, HAVING – fr jako v SQL

4. přednáška – RDF slovníky, Wikidata

- Slovník Dublin Core
 - o Obsahuje základní vlastnosti pro popis knih
 - o Jsou tam dc (pro nás deprecated, ten původní), dcterms (ten co používáme)
 - o Hlavní je třeba – title, publisher, date
- Slovník SKOS (prefix skos)
 - o Poskytuje listy a obecně listy typů atd., což je potřeba unifikovat
 - o Skos:Concept
 - Core class skosu
 - Je to pro basically cokoliv, pro nějaký nápad, jednotku nápadu
 - o Skos:ConceptScheme
 - Taxonomie pro koncepty, tedy vlastně sjednocení více konceptů
 - o Skos:inScheme – říká v jakém schématu je
 - o Skos:topConceptOf – řekne si tím vrcholem jaké struktury je
 - o Skos:prefLabel – human readable reprezentace, pro každý jazyk jedna hodnota
 - o Skos:altLabel – je to alternativní název, tedy může jich být více pro jeden jazyk
 - o Skos:hiddenLabel – pro názvy, které obsahují časté překlepy (tedy často uživatel něco hledá pod špatným názvem, tak sem se dá ta hodnota, aby se to dalo dohledat)
 - o Skos:notation – interní reprezentace pro nějaké hodnoty (výborně by mělo notaci A, tedy A je symbolická hodnota pro známku Výborně)
 - o Skos:collections
 - Skos:collection -> pomocí skos:member má členy
 - Skos:orderedCollection -> pomocí skos:memberList dodám členy v daném pořadí
 - o Skos:mappingRelation
 - Spojuje podobné koncepty napříč různými conceptSchemes

To specify mapping/alignment between schemes.

- skos:mappingRelation
 - o skos:closeMatch (not transitive)
 - skos:exactMatch (transitive)
 - o skos:relatedMatch
 - o skos:broadMatch
 - o skos:narrowMatch

```
<https://data.mvcr.gov.cz/zdroj/ciselniky/pohlavi/položky/zenské> a skos:Concept;
skos:inScheme <https://data.mvcr.gov.cz/zdroj/ciselniky/pohlavi>;
skos:prefLabel "Female"@en, "Ženské"@cs ;
skos:exactMatch <https://data.cssz.cz/resource/ciselniky/ciselnik-pohlavi/2/2009-01-01>,
<http://publications.europa.eu/resource/authority/human-sex/FEMALE> .
```

- GoodRelations

- Slouží pro popis e-shopů

1. Agent - **gr:BusinessEntity**

- a. person or organization

2. Object or service - **gr:ProductOrService**

- a. camcorder, house, car
- b. haircut

3. Promise (offer) - **gr:Offering**

- a. To transfer some rights (ownership, usage, license) on the object or
- b. To provide the service for a certain compensation (money)
- c. Made by the agent and related to the object or service

4. Location - **gr:Location**

- a. From which this offer is available
 - i. store, bus stop, gas station

○

- Gr:openingHoursSpecification

- Specifikuje časové intervaly a linkuje se ke dnům (také jsou v GR)
- Obsahuje i svátky atd., tedy je to list a má více records

- Gr:Offering – nabízí službu nebo produkt, je to ale hlavně pro nějaké zboží

- Gr:priceSpecification – zase list, kde je cena, měna, do kdy platí atd.

- Pro popis produktu nebo služby

- Gr:individual – pro jeden jediný produkt, jedne ntb se sériovým číslem, nebo auto s jedním VINem (každý je unikátní)
- Gr:productOrServiceModel – mluví o věcech jako o sérii (lenovo yoga, asus zenbook atd.)
- Gr:someItems – označuje nějakou podmnožinu z celku, o které platí stejná vlastnost

- Gr:quantitativeValue nebo QualitativeValue – první je pro hodnotové vlastnosti (napětí od do), pak jsou kvalitativní (velikost trička)

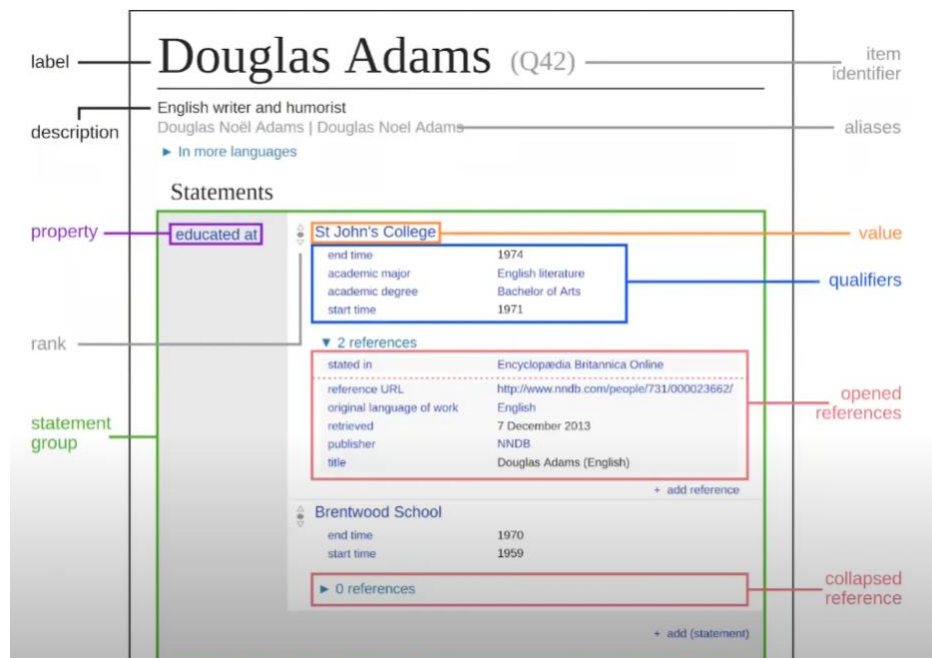
- Schema.org

- Kouká se na existující slovníky a importuje si z nich classy a vydává je v rámci svých služeb

- Wikidata

- Případ užití datové modelu RDF
- Je to komunitně budovaná strukturovaná databáze, stejně jako wikipedia ale wikipedia je na texty ale tohle je vyložené na záznamy v db
- Wikibase je ten sw, který má ta data, běží na něm ta db

Struktura wikidata:



1:05:00 – 1:12:00

Existuje sparql endpoint, pomocí kterého se můžeme dotazovat na wikidata

- Každá položka má svoje id číslo, pomocí kterého se pak dotazujeme
- Tedy položky slovníku pro wikidata jsou vždy nějaké kódy
- Ve zbytku přednášky byl prostě příklad toho, jak se dotazuje nad wikidaty a jak se hodnoty provazují, jak vypadá jejich struktura (od 1:12:00 do konce)

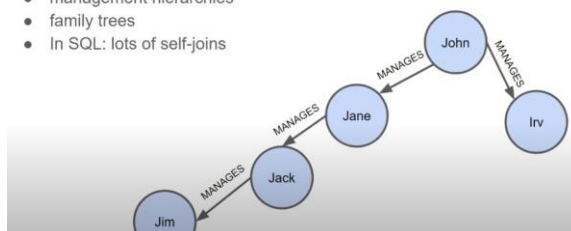
5. přednáška – LPG, Cypher

Grafové a negrafové problémy

- Hodně věcí má přirozeně grafovou strukturu – sítě, kabelové cesty, internetové stránky, sociální sítě, firmy a tok peněz
- Grafy se hodí právě tehdy, když vztahy mezi entitami jsou stejně nebo spíš více důležité než samotné entity
- Grafy se hodí na případ, kde jsou nějaké management hierarchie a rodinné stromy

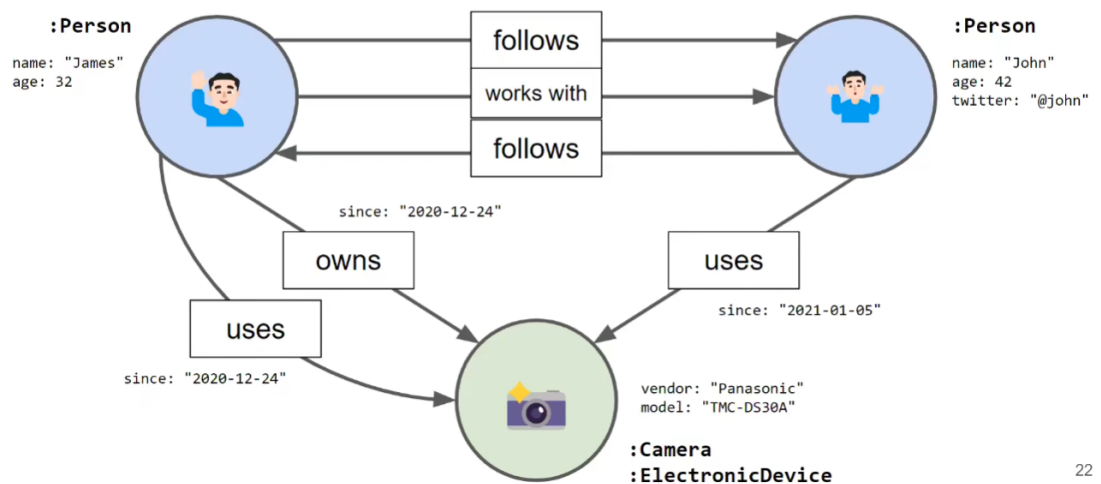
Graph use case: self-referencing

- management hierarchies
- family trees
- In SQL: lots of self-joins



- Dále se grafy hodí, pokud potřebuji graf procházet do hloubky
 - Pokud selže switch, co dalšího selže? Jak moc se zanořit?
- Dále se grafy hodí na klasické grafové problémy, hledání nejkratší cesty apod.
- LPG – labeled property graph
 - Je to grafový datový model
 - Jsou tady jiné nástroje pro reprezentaci grafu na rozdíl od RDF
 - Jsou tam nodes a relationships (orientované hrany, multihrany)
 - Vrcholy u sebe mohou mít seznam hodnot, jsou to vlastně vlastnosti toho nodeu
 - V LPG není žádné schéma
 - Další rozdíl od RDF je ten, že i hrany mohou mít svoje seznamy hodnot (vlastnosti hrany, tedy upřesnění vztahu)

Labeled Property Graph data model



- Další rozdíl je, že hrany ani uzly nejsou identifikovány žádným IRI, jsou vlastně jen pojmenovány
- Uzly mohou mít labels uvedené dvojtečkou
- Velmi dobře se mapují do angličtiny, hezky se to čte
- Cypher
 - Je to dotazovací jazyk nad LPG
 - Umožňuje CRUD operace
 - LPG i cypher (oboje od neo4j) jsou vlastně velmi closed, ale cypher je již teď open (jeho specifikace)

Cypher - AsciiArt

Nodes

() (p)

Labels start with : and represent types

(p:Person:Mammal)

Nodes can have properties

(p:Person {name: 'John'})

Relationships

--> -[f:FOLLOWS]->

Direction of relationship can go both ways

(p1)-[:FOLLOWS]->(p2) or (p1)-[:FOLLOWS]-(p2)

Relationships can have properties too

-[:OWNS {since: '2021-02-21'}]->

- Match – co hledám
- Where – dám podmínku, která musí platit pro nějakou vlastnost
- Set – přidám tak nějaký key-value pair, nemusel tam klidně být
- Constraint – jelikož tam nejsou žádná schémata, tak mohou často vznikat duplicitní data. Mohu tedy vytvářet constraints:

```
(CREATE CONSTRAINT ON (p:Person)
  ASSERT p.name IS UNIQUE)
```

- MERGE – zkusí najít něco, pokud to tam je, nic se nestane ale pokud tam není, tak se udělá
- MERGE + ON CREATE – při vytvoření něco nastavím
- Jsou tam i grafové algoritmy – shortest path

Cypher style guide

Node labels

- Case sensitive
- `UpperCamelCase`
i.e. beginning with an upper-case character
- `:VehicleOwner` rather than `:vehicle_owner` etc.

Relationship types

- Case sensitive
- `SCREAMING_SNAKE_CASE`, i.e. upper-case, using underscore to separate words
- `:OWNS_VEHICLE` rather than `:ownsVehicle` etc.

Property keys

- Case sensitive
- `lowerCamelCase`
- `twitterHandle` rather than `TwitterHandle`

Cypher keywords

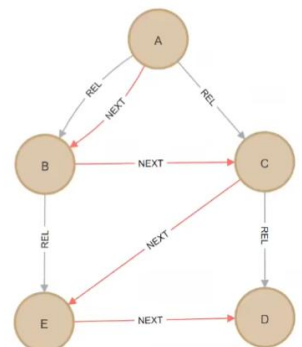
- Case insensitive
- However, recommended to use upper-case
- `MATCH` rather than `MaTch`

- RETURN klauzule rovnou grupuje, tedy skoro není potřeba explicitní GROUP BY
- WHERE – klasika pro filtrování dat
- Null je v cypheru spíše jak missing value, ne vyloženě jako hodnota „nic“ uložena v db
- WITH – manipuluje s nějakou hodnotu předtím, než je předána dalším částem query
- Grafových algoritmů je tam dost a v tom se to fr vyplatí používat

Graph algorithms, Machine-learning support, Graph administration

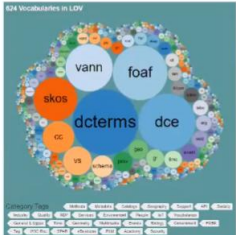
- Centrality
 - PageRank, Degree Centrality, ...
- Community detection
 - Weakly Connected Components, ...
- Similarity
 - Node similarity, ...
- Path finding
 - Dijkstra, BFS, DFS, ...
- Node embeddings
 - Node2Vec, ...

```
MATCH (source:Node{name:'A'})
CALL gds.bfs.stream('myGraph', {
  sourceNode: source
})
YIELD path
RETURN path
```


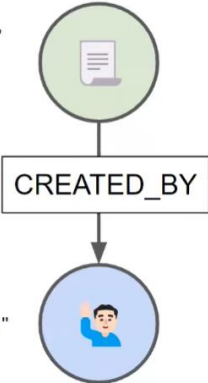


NPRG036 – Datové formáty

- GQL – graph query language, mělo by to být jako ISO standard, ale stále nic nevyšlo, bude to postaveno na openCypher
- RDF vs LPG

Resource Description Framework (RDF)	Labeled Property Graph (LPG)
Made for the Web, distributed data	Made for centralized graph data
<ul style="list-style-type: none">• global identification: IRIs for everything• globally reused RDF vocabularies<ul style="list-style-type: none">○ focus on meaning of data• focused on linking data from various publishers	<ul style="list-style-type: none">• local node labels and edge types• every database instance uses different relationship types and node labels• better at evaluating graph algorithms<ul style="list-style-type: none">○ e.g shortest path
	

-
- Rdf je hlavně pro data na webu, vhodná pro hodně různorodá data a velké množství
- LPG chce hlavně grafové algoritmy, na centralizovaná data
- Jak říct nějakou další informaci o nějaké trojici?
 - V LPG dám jako seznam hodnot k hraně
 - V RDF si pojmenuji danou trojici a pak o ní něco řeknu na základě toho nového jména (to jméno reprezentuje tu trojici)

Resource Description Framework (RDF)	Labeled Property Graph (LPG)
<pre>my:index.html my:createdBy "Jakub Klímek" .</pre>  <pre>_:triple1 a rdf:Statement . _:triple1 rdf:subject my:index.html . _:triple1 rdf:predicate my:createdBy . _:triple1 rdf:object "Jakub Klímek" . _:triple1 dcterms:source "https://x.y.z"^^xsd:anyURI . _:triple1 dcterms:created "2020-04-23"^^xsd:date .</pre>	<pre>name: "my:index.html"</pre>  <pre>cameFrom: "https://x.y.z" scrapedOn: "2020-04-23"</pre> <pre>name: "Jakub Klímek"</pre> <ul style="list-style-type: none">▪ Kvůli tomuhle vzniklo RDF* - RDF Star

RDF-star - RDF*

Emerging technique, see the [latest editor's draft](#).

Allows RDF statements to appear as subjects and objects of other RDF statements.

Based on RDF reification, but weaker.

The triple in the subject or object position is enclosed in `<< >>`, called **"quoted triple"**.

```
#Bob states: there is someone named Alice
#There is someone named Alice, stated Bob.
<< _:a :name "Alice" >> :statedBy :bob.
```

```
#Employee22 claims that employee38's job title is
Assistant designer
:employee38 :familyName "Smith" .
:employee22 :claims
<< :employee38 :jobTitle "Assistant Designer" >> .
```



6. přednáška – XML, XML Schema

- XML – historie
 - Vznikalo jako označovaný souvislý text, kde tagy označovaly místa pro nějaké atributy -> document-oriented xml
 - Xml je hierarchický model (stromový)
- My pracujeme s datově orientovaný xml (data-oriented xml)
- XML = eXtensible Markup Language
- Prolog – je to první řádek xml dokumentu
 - Obsahuje kódování a verzi xml (typický 1.0)
 - Dnes xml používá utf-8
- Mixed content – obsahuje prvky xml ale i kusy textu
- Xml dokumuent je dobře zformovaný, pokud splňuje pravidla xml syntax rules (nejde o schéma)
- Xml namespacey – names space se dává do elementu `<h:table>`
 - Definují se v kořeni, aby pak byly použitelné v celém dokumentu

```
<root xmlns:h="http://www.w3.org/TR/html4/"
      xmlns:f="https://www.w3schools.com/furniture">
  <h:table>
    <h:tr>
      <h:td>Apples</h:td>
      <h:td>Bananas</h:td>
    </h:tr>
  </h:table>
  <f:table>
    <f:name>African Coffee Table</f:name>
    <f:width>80</f:width>
    <f:length>120</f:length>
  </f:table>
</root>
```

-
- Cdata sekce
 - Jsou to znaková data, je tam jediný rozdíl, že speciální znaky v cdata není třeba escapovat

XML CDATA sections

```
<?xml version="1.0" encoding="UTF-8"?>
<elementA>
  <![CDATA[<greeting>Hello, world!</greeting>]]>
</elementA>
```

Everything between <![CDATA[and]]> is treated as string

-
- Má to zase xml:lang atribut, což je jako language tag v RDF
- Xml processing instruction – PI
 - <?xml-stylesheet type="text/xsl" href="style.xsl"?>
 - Vypadá to jako prolog, ale je to úplně samostatný a nezávislý řádek
- DOM – document object model
 - Způsob processování xml data v aplikaci
 - Funguje tak, že se celé xml načte do paměti
 - Používá se pro html, je to dobré na random access
- SAX – simple api for xml
 - Sax parser je vhodnější pro velká xml, volá to api při procházení
 - Parsuje element po elementu, tedy ale není vhodný na random access, tedy chci se dokázat na random element
- STAX – streaming api for xml
 - Stejně výhody i nevýhody jako sax
 - Rozdíl je, že v staxu člověk kontroluje, jak jde parsování
 - V saxu parser jede a volá váš kód, který s daty něco dělá
 - Ve staxu kód volá parser (zpracuj další element), tedy když chci skončit tak mohu, ne jako v saxu, kde prostě parser jede do konce
- XML je jako hostitelský formát pro další formáty jako SVG
- XML lze validovat (pomocí XML schema jsou ale i jiné)
- Dají se dotazovat (XPath)
- Dá se transformovat do jiných formátů (XLT)
- Existuje velmi stará serializace RDF/XML
- SVG – vektorová logika pomocí xml jako hostitelského formátu

- OOXML – open office xml, jako je i v MS Word atd, kde je docx, vše jsou to vlastně zazipovaná xml
- Atom, RSS – formát pro posílání novinek na webu
- Xml schema

- Zase používá xml jako hostitelský formát

XSD and XML documents / instances

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  ... <!-- XML schema definition --> ...
</xs:schema>
```

```
<?xml version="1.0" encoding="utf-8"?>
<root_element_of_XML_document
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="schema2.xsd">
  ... <!-- XML document --> ...
</root_element_of_XML_document>
```

Link to XML Schema

-
- Xml dokument je validní, pokud má xml schema a je vůči němu validní
- Xml schema 1.1 – obsahuje simple typy, bool, anyURI, date, time, dateTime (prakticky stejné jako v RDF a dalších formátech)
- Xml schéma definuje datové typy, elementy, jejich skupiny, a i atributy elementů (skupiny mohou být i pro atributy)
- Element simple type
 - Ověřuje to samotný typ, můžeme je omezovat (min, max atd.),
 - Typ si můžeme i pojmenovat a pak reusovat
 - Můžu udělat element jako list typů
 - Pak to může být union, kde může v jednom elementu být třeba boolOrInteger
- Element complex type
 - Sám v sobě ještě může obsahovat simple type, a navíc ještě pro něj mohu definovat atributy (nelze pro simple types)
 - Complex type může mít i complex content, kde mohu definovat, jaké daný element může mít podelementy
 - Choice vs sequence vs all
 - Choice je skutečně výběr, může tam být libovolné z toho
 - Sequence je zase že tam musí být vše v tom pořadí, ve kterém to je ve schématu, jinak to není validní
 - All – všechny elementy tam musí být, ale nezáleží na pořadí, na rozdíl od sequence
- Ref – atribut pomocí kterého se odkáží na již definované schéma elementu

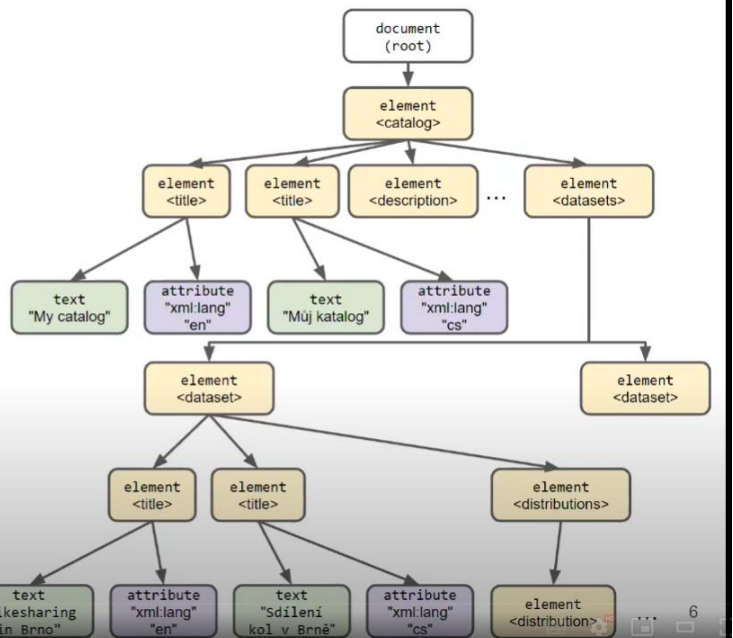
- Namespaces
 - Používám-li namespacey, tak chéma je schemaLocation atribut
 - Pokud je nepoužívám, tak schéma je noNamespaceSchemaLocation

7. přednáška – XPath, XSLT

- XPath – dotazovací jazyk nad XML
- Zavedeme si datový model pro xml, aby data měla nějakou strukturu na dotazování
 - o XPath data model – založeno na xml information set, díky tomu je podpora pro XPath dotazy a XSLT

XPath Data Model

```
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
  <title xml:lang="en">My catalog</title>
  <title xml:lang="cs">Můj katalog</title>
  <description xml:lang="en">This is my dummy catalog</description>
  <description xml:lang="cs">Toto je můj falešný katalog</description>
  <contact-point>
    <name xml:lang="en">John Doe</name>
    <e-mail mailto:john@doe.org</e-mail>
  </contact-point>
  <datasets>
    <dataset>
      <title xml:lang="en">Bikesharing in Brno</title>
      <title xml:lang="cs">Sdílení kol v Brně</title>
      <distribution>
        <media-type>application/xml</media-type>
        <downloadURL>http://brno.cz/myfile.xml</downloadURL>
      </distribution>
      <accessService>
        <endpointURL>https://brno.cz/myAPI</endpointURL>
        <title xml:lang="en">My API</title>
      </accessService>
    </dataset>
    <dataset>
      <title xml:lang="en">Bikesharing in Prague</title>
      <title xml:lang="cs">Sdílení kol v Praze</title>
      <distribution>
        <media-type>text/csv</media-type>
        <downloadURL>http://praha.eu/myfile.csv</downloadURL>
      </distribution>
    </dataset>
  </datasets>
</catalog>
```

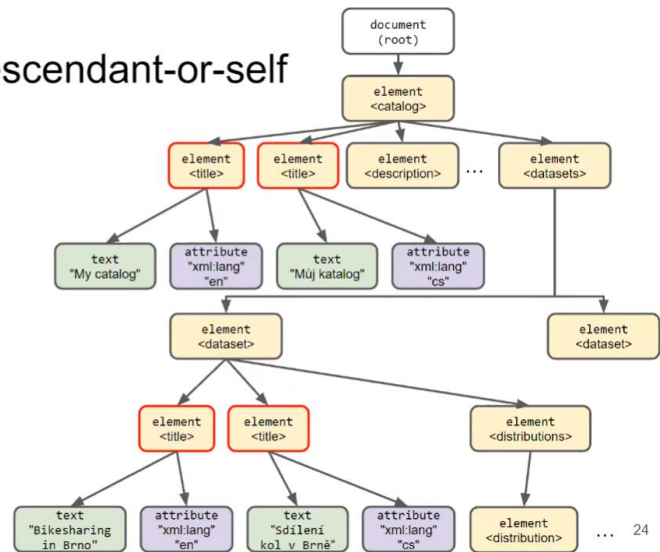


- Text() – funkce která mi uvnitř dotazu vytáhne textový obsah z daného xml elementu
- @xml:lang – zavináčem se ptám na atributy, jejich obsah, hodnotu
- Predikát – [predicate] – sem dám nějakou logical expression jako where, nějakou podmínku, podle které hledám
- Axis – jsou to skutečně osy podle toho, jakým způsobem/směrem chci hledat
 - o Pokud použiju *, znamená to všechny elementy splňující danou osu
 - o Child osa – je to default hodnota, vrátí všechny elementy
 - o Descendant osa – jdu do všech potomků, do jakékoliv hloubky
 - o Attribute osa – má syntaktickou zkratku @, jak zmíněno výše (vlastně low-level implementace zavináče)
 - o Preceding sibling osa – bere předchozí sourozence

- Descendant-or-self osa – self má každá osa, vlastně includeuje node, ze které se ptám

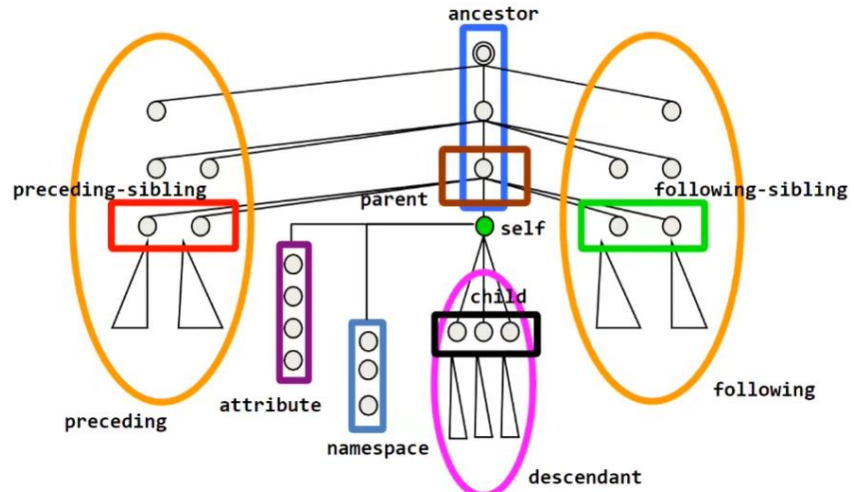
XPath - axes - descendant-or-self

`/catalog/descendant-or-self::title`
`/catalog//title`



- - Tedy tady se to kouká i na přímé descendants katalogu
 - Syntaktická zkratka je //
- Self osa – obsahuje element, ve kterém zrovna jsem (zkratka .)
- Parent osa – obsahuje rodiče elementu, ve kterém zrovna jsem (zkratka ..)

XPath - all axes



-
- Funkce – vrací něco z elementu
 - Name() – vrací název elementu
 - Position() – vrací pozici sebe a sourozenců v daném elementu, lze to indexovat jako list (chci druhý element v rodiči)
 - Last() – vrací počet elementů v uzlu
- XSL Transformation – XSLT
 - XSLT jazyk má zase jako hostitelský jazyk XML
 - Input je 1 a více XML dokumentů
 - Output jsou text files jako HTML, XML, RDF Turtle – fr jakýkoliv text

- Xslt stylesheet – xml dokument, který obsahuje šablony (xml templates) a pak se to dá do xslt procesoru, který matchuje xml dokument na templaty a transformuje je

XSLT - empty stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">

  <xsl:output method="html" encoding="UTF-8" indent="yes" />

</xsl:stylesheet>
```

version attribute - version of XSLT used

xsl:output - specifies the output behavior of the XSLT processor

- **method**
 - **html, xhtml, xml** - produces well-formed documents
 - **text** - pure text output
- **indent**
 - **yes** - generates correct indentation for xml, html
 - **no** - only explicitly generated whitespace included in output

- - Match atribut – má xpath expressions kterou bude matchovat
 - Xsl:template – blueprint na to, jak má být něco transformováno
 - Xsl:value-of – dostanu tím hodnotu xpath dotazu do výstupu
 - Xsl:apply-templates – aplikuje další šablony jako děti
 - Implicitní šablony – má tedy nějaké předdefinované šablony, např. šablony matchující všechny texty, elementy, atributy a vypisuje jejich basic obsah
- ## XSLT - named templates and parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">

  <xsl:template match="/">
    <html>
      <xsl:call-template name="processTitle">
        <xsl:with-param name="element">title</xsl:with-param>
      </xsl:call-template>
    </html>
  </xsl:template>

  <xsl:template match="processTitle">
    <xsl:with-param name="element">h1</xsl:with-param>
    <xsl:with-param name="lang">en</xsl:with-param>
    </xsl:template>

  <xsl:template match="dataset/dataset">
    <table>
      <xsl:call-template name="processTitle">
        <xsl:with-param name="element">h2</xsl:with-param>
        <xsl:with-param name="lang">en</xsl:with-param>
      </xsl:call-template>
      <xsl:apply-templates select="dataset/dataset"/>
    </table>
  </xsl:template>

  <xsl:template match="dataset">
    <xsl:call-template name="processTitle">
      <xsl:with-param name="element">h3</xsl:with-param>
      <xsl:with-param name="lang">en</xsl:with-param>
    </xsl:call-template>
    <xsl:apply-templates select="dataset/dataset">
      <xsl:with-param name="lang">en</xsl:with-param>
    </xsl:apply-templates>
  </xsl:template>

  <xsl:template match="text">
    <xsl:element name="p">
      <xsl:value-of select="."/>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

Named templates

- **name** attribute instead of **match** attribute
- **accept parameters**
 - **xsl:param** - definition in named template
 - **\$variable** - access to variable value in XPath
 - **{ \$variable }** - access to variable value elsewhere
- called using **xsl:call-template**
 - does not change the currently processed node set
 - **xsl:with-param** - values passed when calling

xsl:element

- creates an element on the output
- name can be constant or **{ \$variable }**

-
- Existují i globální proměnné, if podmínky atd.
 - např. pro nastavení jazyka pro celý dokument

XSLT - global variables, modes, if

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:fn="http://www.w3.org/2005/xpath-functions">

  <xsl:variable name="lang">en</xsl:variable>

  <xsl:template match="/">
    <html>
      <xsl:apply-templates mode="head"/>
      <xsl:apply-templates mode="catalog"/>
    </html>
  </xsl:template>

  <xsl:template match="dataset" mode="head">
    <xsl:apply-templates mode="catalog">
      <xsl:with-param name="lang">en</xsl:with-param>
    </xsl:apply-templates>
  </xsl:template>

  <xsl:template match="title" mode="head">
    <xsl:if test="fn:lang($lang)">
      <xsl:element name="title">
        <xsl:value-of select="."/>
      </xsl:element>
    </xsl:if>
  </xsl:template>

  <xsl:template match="title" mode="catalog">
    <xsl:if test="fn:lang($lang)">
      <xsl:element name="h1">
        <xsl:value-of select="."/>
      </xsl:element>
    </xsl:if>
  </xsl:template>

  <xsl:template match="text" mode="all">
    <xsl:element name="p">
      <xsl:value-of select="."/>
    </xsl:element>
  </xsl:template>

</xsl:stylesheet>
```

Global variable

- defined in the **xsl:stylesheet** root element using **xsl:variable**
- accessible in the whole stylesheet
 - e.g. **\$lang**

Mode

- ability to process the same nodes in different ways
 - different templates with the same **match**
- specified in **xsl:apply-templates**
- used in unnamed **xsl:template**
 - **#all** matches all modes

-
- Je tam switch a foreach – často se špatně používají místo apply-templates, což by se nemělo (jde to, ale má to dopad na výkon atd.)

8. přednáška – JSON, JSON Schema, JSON-LD

- JSON – javaScript object notation
- Používá se hlavně utf-8
- Json number – podpora pouze pro desítkové, žádné hexadecimální atd., používá se tečka pro desetinnou hodnotu
- Json string – klasika, speciální znaky je třeba escapovat (new lines, uvozovky atd.)
- Whitespacey – pro indentaci
- Json value – může to být string, number, bool, null, object, array
- Typicky se používá na web apis, je to datový standard pro české datové sady, používá se i pro databáze (document dbs)
- Jsonl – json lines, kde na každém samostatné řádku je validní json
- Json pointer – funguje to jako cesta, která ukazuje dovnitř json souboru
- JSON schema – slovník pomocí
 - o Nemá žádný standard, ale hodně lidí to používá

JSON Schema

<https://json-schema.org/>
Proposed IETF standard

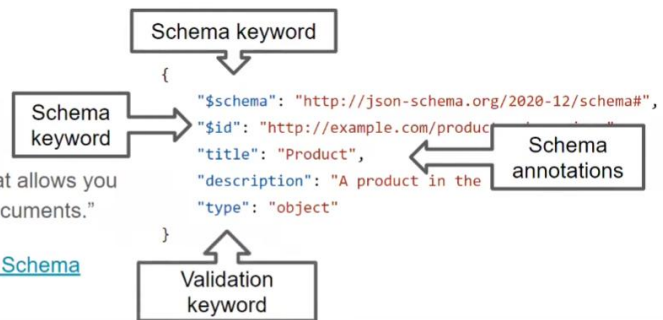
"JSON Schema is a vocabulary that allows you to **annotate** and **validate** JSON documents."

Online book: [Understanding JSON Schema](#)

First draft: 2009

Still under development

```
{
  "productId": 1,
  "productName": "A green door",
  "price": 12.5,
  "tags": [ "home", "green" ]
}
```



✓ No errors found. JSON validates against the schema.

- Atributy pro json schema
 - o Properties – definuje jednotlivé atributy (názvy hodnot, key-value pairs) daného json objektu nebo arraye a říká jeho typ, jestli je required a description
 - o Jsou tam i speciální hodnoty pro numerické hodnoty

Validation keywords for numeric instances

- multipleOf
- maximum, exclusiveMaximum
- minimum, exclusiveMinimum

- JSON arrays

JSON Schema - JSON arrays - list validation

```
{
  "productId": 1,
  "productName": "A green door",
  "price": 12.5,
  "tags": [ "home", "green" ]
}
```

```
...
"properties": {
  ...
  "tags": {
    "description": "Tags for the product",
    "type": "array",
    "items": {
      "type": "string"
    },
    "minItems": 1,
    "uniqueItems": true
  },
  ...
}
```

Validation keywords for arrays - list validation

- items, maxItems, minItems
- contains, maxContains, minContains
- uniqueItems

JSON Schema - JSON arrays - tuple validation

Each item in the array has different schema

Positions are meaningful

```
{
  "type": "array",
  "items": [
    { "type": "number" },
    { "type": "string" },
    {
      "type": "string",
      "enum": [ "Street", "Avenue", "Boulevard" ]
    },
    {
      "type": "string",
      "enum": [ "NW", "NE", "SW", "SE" ]
    }
  ],
  "additionalItems": { "type": "number" }
}
```

✓ [1600, "Pennsylvania", "Avenue", "NW"]

✗ [24, "Sussex", "Drive"]

✗ ["Palais de l'Élysée"]

✓ [10, "Downing", "Street"]

✗ [1600, "Pennsylvania", "Avenue", "NW", "Washington"]

- JSON objects

JSON Schema - nested JSON objects

```
{
  "productId": 1,
  "productName": "A green door",
  "price": 12.5,
  "tags": [ "home", "green" ],
  "dimensions": {
    "length": 7.0,
    "width": 12.0,
    "height": 9.5
  }
}
```

```
...
"properties": {
  ...
  "dimensions": {
    "type": "object",
    "properties": {
      "length": { "type": "number" },
      "width": { "type": "number" },
      "height": { "type": "number" }
    },
    "required": [ "length", "width", "height" ]
  },
  ...
}
```

- Json schéma lze specifikovat i z jiného souboru pomocí \$ref

JSON Schema - external schema

```
{
  "$id": "https://example.com/geographical-location.schema.json",
  "$schema": "http://json-schema.org/2020-12/schema#",
  "title": "Longitude and Latitude",
  "required": [ "latitude", "longitude" ],
  "type": "object",
  "properties": {
    "latitude": {
      "type": "number",
      "minimum": -90,
      "maximum": 90
    },
    "longitude": {
      "type": "number",
      "minimum": -180,
      "maximum": 180
    }
  }
}
```

```
...
"properties": {
  ...
  "warehouseLocation": {
    "description": "Coordinates of the warehouse ...",
    "$ref": "https://example.com/geographical-location.schema.json"
  },
  ...
}
```

- I json strings mají constraints – minLength, maxLength, pattern (je to regex)
- Json formats

JSON Schema - formats

built-in formats for strings

- date-time, date, time, duration
- email, idn-email
- hostname, idn-hostname
- ipv4, ipv6

- uri, uri-reference
- iri, iri-reference
- uuid
- uri-template
- json-pointer, relative-json-pointer
- regex

```
"typ_turistického_cile": {
  "type": "string",
  "format": "iri",
  "pattern": "^https\\:\\\\data\\.mvr\\.gov\\.cz\\zdroj\\ciselniky\\typy-turistickych-cilu\\polozky\\.*$",
  "title": "Typ turistického cíle",
  "examples": [
    "https://data.mvr.gov.cz/zdroj/ciselniky/typy-turistickych-cilu/polozky/prirodni"
  ]
}
```

JSON Schema - combining schemas

Schemas can be combined

- anyOf
 - valid against at least one schema
- allOf
 - valid against all schemas
- oneOf
 - valid against exactly one schema
- not
 - valid against none of the schemas

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

Impossible

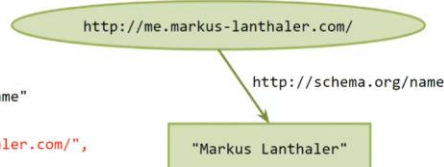
```
"anyOf": [
  { "type": "string", "maxLength": 5 },
  { "type": "number", "minimum": 0 }
]
```

```
"not": { "type": "string" }
```

-
- Je tam i podpora pro non-json hodnoty – contentType, contentEncoding
 - Třeba type: string, contentType: „text/html“
- JSON-LD = json for linked data
 - Jazyk, díky kterému mohu json udělat interpretovatelný jako RDF model
 - Vyznačuje se tím, že začíná @

JSON-LD - subject identifier (IRI)

```
{
  "@context": {
    ...
    "name": "http://schema.org/name"
  },
  "@id": "http://me.markus-lanthaler.com/",
  "name": "Markus Lanthaler",
  ...
}
```



- }
- @type – udává typ pro predikát
- Contextu mohu zadefinovat base a všechny iri pak budou relativní vůči tomu base
- Pokud nastavím v kontextu hodnotu jako null, tak se jí pak zbavím při vyrenderování RDF

- V contextu is také mohu definovat prefixy – foaf: „http://foaf.org/“
- V jsonu se používají xml schema datové typy
- Mohu si definovat context uvnitř nějakých objektů – scoped context
- Jazyky – v contextu lze říct v jakém jazyku je celý dokument pomocí @language
 - Pak pro atributy mohu dát do názvu názvy v různých jazycích:
 - Name_en: „dude“
 - Name_cs: „kámo“
 - A to se přeloží jako string s language tagem
- Můžeme to také aliasovat v context – url: @id -> @id si přejmenuji na url
- Context mohu přidat i jako externí, url na daný soubor s contextem

9. přednáška – CSV, CSV on the Web

- Relační datový model
 - Klasika o relačním datovém modelu (tabulka = relace atd.)
- DSV – delimiter separated values
 - Předchůdce csv, je to z unixového světa
- TSV – tab separated values
 - Řádky jsou oddělené new lines
 - Má hlavičku, taby nesmí být ve values
- CSV – Comma separated values
 - Specifikace CSV je RFC 4180
 - Výchozí kódování je US-ASCII (dnes už samozřejmě utf-8)
 - Escape character jsou uvozovky “
 - Line ending je CRLF vždy
- Datové typy pro csv jsou zase z XML schema
 - Xsd:boolean, xsd:integer atd. (stejně jako všech ostatní formátů)
- Header být nemusí, je jen doporučený
- Missing values jsou skutečně, že tam nic není. Null je vlastně hodnota s hodnotou null
- Pokud mám třeba váhu, tak je nejlepší to rozdělit na 2 sloupce, hodnota a jednotka
- V CSV by se neměli dávat součty, skutečně by to měli být values jen podle headeru
- Pokud máme nějaké url, které identifikuje csv soubor, tak pomocí fragmentů v url se mohou podívat na jednotlivé buňky v csv
(<http://example.com/data.csv#cell=4,1>)

- CSV on the Web (= CSVW)
 - o Anotace csv tabulek se dělá pomocí json-ld
 - o Používá se pro anotaci, validaci a transformaci na jiné formáty

CSVW - JSON-LD descriptor - Table schema

Table schema describes columns, rows and cells

```
{
  "@context": "http://www.w3.org/ns/csvw",
  "@type": "Table",
  "@id": "https://example.org/table1",
  "url": "https://example.org/table1.csv",
  "tableSchema": {
    "columns": [
      {
        "titles": "airport",
        "dc:description": "An identifier for the airport.",
        "datatype": "string",
        "required": true
      },
      {
        "titles": "continent",
        "dc:description": "The continent the airport is on.",
        "datatype": "string",
        "lang": "en",
        "required": true
      }
    ]
  }
}
```

airport	continent
PRG	Europe
DXB	Asia

- Pro typy z xml jsou i zkratky – datetime = xsd:dateTime, number = xsd:double
- Datové typy mohou být upravené (regex, délka, max/min hodnota atd.)
 - o dataType: integer
 - o minimum: 1
 - o maximum: 10
- name – je to název pro sloupec, pro jeho referenci
- titles – skutečné možné názvy, mohou tam uvádět i názvy pro jiné jazyky
- primaryKey – tam právě použijí hodnotu v name, ne v titles
- foreignKey – také použijí name, referenci sloupce v jiné tabulce
- CSV dialekty
 - o Mohu si upravit nastavení csv, předefinovat encoding, konce řádků atd.
 - o Podpora pro seznamy v hodnotách – jedná řádku obsahuje informaci odkud kam místo toho, aby to bylo rozdělené do 2 sloupců
 - o Mohu si i předefinovat hodnotu null, vlastní formátování času
- JSON-LD descriptor pro nějaký csv soubor se většinou jmenuje:
 - o {název_csv}-metadata.json
- Další funkcionality csv on the web je i generování RDF ze CSV
 - o {var} – do těch závorek přijde hodnota dané proměnné
 - {+var} – může obsahovat procentuálně encoded znaky
 - {#var} – výsledek může mít prefix #
 - o aboutUrl – uri pro rdf subjekt
 - o propertUrl – uri pro rdf predikát
 - o valueUrl – uri pro rdf objekt
 - o virtual – mohou tak označit sloupec, který neexistuje v tom csv souboru
 - takový sloupec lze třeba použít pro generování dalších trojic, které bychom nemohli vlastně navěsit na existující sloupce v CSV

CSVW - the rest - transformations

transformations

- on Table groups and tables
- define how tabular data can be transformed into another format using a script or template
- **url**
 - URL of the script or template
- **scriptFormat**
 - If defined, IANA media type, if not, any URL
- **targetFormat**
 - If defined, IANA media type, if not, any URL
- **source**
 - specifies standard transformation prior to transformation using script or template
 - **json**, **rdf**, **null**
- **titles**

```
{
  "@context": "http://www.w3.org/ns/csvw",
  "@type": "Table",
  "@id": "https://example.org/table10",
  "url": "https://example.org/table10.csv",
  "transformations": [
    {
      "@type": "Template",
      "url": "templates/ical.txt",
      "titles": "iCalendar",
      "targetFormat":
        "http://www.iana.org/assignments/media-types/text/calendar",
      "scriptFormat": "https://mustache.github.io/",
      "source": "json"
    }
  ]
}
```

72

- Validace oficiální není, jsou na to offline tools (RDF::Tabular)

10. přednáška – Geodata

- Spatial data
 - Data to answer spatial questions – how far is it, which way to go
- It has the ISO / TC 211 standard
- Implicit geodata
 - Something that is measurable – coordinates, distance, directions
- Explicit geodata
 - Local names – reference, address, geographical name
- Vector vs raster representation
 - Most geodata is stored using the vector representation
 - Paper map is like the raster representation
 - Vector data is faster to server, raster representation provides pixels
 - Vector representation is much more precise
- There are many geometry objects – points, multipoints, lines, polygons, surface
- Points – restaurants
- Multipoints – all public transportation stations in Dejvice
- Line – D1
- Polygon – the area of České Budějovice (borders)
- Geometry representation – standard
 - WKT – well-known text
 - It has a binary version, it provides a set of coordinates

Well-Known Text (WKT)

```
POINT(50.056 14.434)
■ LINESTRING(50.056 14.434, 50.064 14.442, 50.042 14.445)
```

- GML – geography markup language

- XML is its host language

Geography Markup Language (GML)

```
<gml:Point srsName="http://opengis.net/def/crs/EPSSG/0/4326" srsDimension="2">/
  <gml:pos>50.056 14.434</gml:pos>
</gml:Point>

<gml:Curve srsName="http://opengis.net/def/crs/EPSSG/0/5514" srsDimension="2">
  <gml:segments>
    <gml:LineStringSegment>
      <gml:posList>-641126.76 -1093821.18 -641119.35 -1093831.05
        -641109.75 -1093844.44</gml:posList>
    </gml:LineStringSegment>
  </gml:segments>
</gml:Curve>
```

- Geometry representation – format based

Data format	Geometry representation
GML	GML
GeoJSON	geojson
Shapefile	binary
GeoPackage	SQLite
CSV	any
GeoSPARQL	GML/WKT

- GML

- It has an open specification, based on XML
 - It is very widely used

- GeoJSON

- It is based on json
 - Does not support other coordinate reference systems than wgs-84
 - Has its own geometry representation
 - All the geometry object supported

GeoJSON

```
{
  "geometry": {
    "coordinates": [
      14.419134,
      50.090122
    ],
    "type": "Point"
  },
  "crs": {
    "type": "name",
    "properties": {
      "name": "urn:ogc:def:crs:EPSSG::4326"
    }
  },
  "properties": {
    "cislo_orientacni": "22",
    "cislo_popisne": "128",
    "druh_mista": "RESTAURAČNÍ ZAHŘÁDKY",
    "druh_zbozi": "",
    "mome": "Praha 1",
    "ulice": "Pařížská"
  },
  "type": "Feature"
}
```

- ShapeFile

- The format specification is not opened
 - Native format for GIS in the Czech Republic
 - Consists of multiple files
 - Restricted number of chars per column name

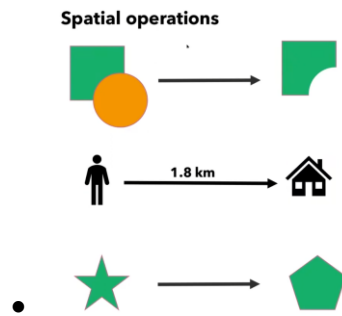
Shapefile



	NAME	DATASOURCE	CRS	WKT	Geometry Type	Field 1	Field 2	Field 3
1	hachovka	6280000	WGS 1984	POINT	Point			
2	hachovka	6280000	WGS 1984	POINT	Point			
3	hachovka	6280000	WGS 1984	POINT	Point			
4	hachovka	6280000	WGS 1984	POINT	Point			
5	hachovka	6280000	WGS 1984	POINT	Point			
6	hachovka	6280000	WGS 1984	POINT	Point			
7	hachovka	6280000	WGS 1984	POINT	Point			
8	hachovka	6280000	WGS 1984	POINT	Point			
9	hachovka	6280000	WGS 1984	POINT	Point			
10	hachovka	6280000	WGS 1984	POINT	Point			
11	hachovka	6280000	WGS 1984	POINT	Point			
12	hachovka	6280000	WGS 1984	POINT	Point			
13	hachovka	6280000	WGS 1984	POINT	Point			
14	hachovka	6280000	WGS 1984	POINT	Point			
15	hachovka	6280000	WGS 1984	POINT	Point			
16	hachovka	6280000	WGS 1984	POINT	Point			

- OGC GeoPackage
 - It is basically an sqlite db
 - Supports both simple and complex geometry structures
 - It is the fastest and the most lightweight way to store geodata
 - It is for both vector and raster data
 - CSV
 - Very easy, MS Excel friendly
 - It does not provide the recommended geometry
 - There are commas in the object representation so it needs to be escaped (using quotes in CSV ofc)
- Spatial linked data formats
 - Geo WGS-82
 - It can only represent points using RDF
 - It works very good but only for points
 - GeoSPARQL
 - It has a query language for spatial operations
 - It has all the geometry objects it needs
 - It is complicated for beginners and very OP for simple tasks
 - GeoJSON-LD
 - Just like normal json-ld, it is about adding context but this for geojson data
- Spatial relations and operations
 - Spatial relations – it is a relation between at least 2 objects and it is usually based on location or shape
 - TOPOLOGICAL – it is represented as a function which returns bool – within(o1, o2), touches(o1, o2) and so on
 - DIRECTIONAL – left or right compared to one stationary object
 - DISTANCE – how are some object from each other
 - TEMPORAL – also accounts for time
 - Spatial operations – ways of analyzing data a certain way
 - Types:
 - Buffer – extend the are of an object so that each point of the object is the same distance from the border of the area
 - Union, difference, interesection – classic

- Clip, distance, convex hull – respective order



- GIS SW and spatial libs
 - o QGIS – open source for all the shown geodata formats (except for the linked ones)
 - o PostGIS – spatial extension for PostgreSQL
 - o ArcGIS – large commercial project
 - o Leaflet – lightweight js libs for maps
 - o Openlayers – js api for maps
 - o Mapserver and geoserver – very heavy solutions and store in spatial dbs on the server
 - Usually used for serving geodata

11. přednáška – key-value formáty

- Používají se hodně pro konfigurační files
- .properties file
 - o Čistě pro nejhorší jazyk na světě (java), nikde jinde
 - o Má java-specific encoding (iso-8859-1)
 - o Je to organizované jako hashtable
- Ini file
 - o Vznikl ofiko v ms dos, dále pak pro microsoft windows
 - o Má to stejný encoding jako .properties
 - o Velmi jednoduchý, nemá ofiko specifikaci
 - o Dělí se na sekce (db, owner) a ty mají key-value hodnoty
- TOML
 - o Vypadá velmi podobně jako ini file
 - o Encoding je již v unicode
 - o Má normální typy ve values jako stringy, inty, ...
 - o Jako klíče jsou bare-keys (ahoj = „ahoj“), quoted („ahoj“ = „ahoj“), dotted (dotted.key = „ahoj“)
 - Dotted key je vlastně 2 zase hashtable (access na atribut)
 - o Umožňuje escaping backslashem, multiline values (pomocí “””)

- Pro čísla je podpora pro nekonec, _ pro oddělování řádů
- Umí klasicky typy date, time, dateTime
- Podpora pro arraye, hashtables (jako v jsonu, jen to má jinou notaci)
- Array of tables

TOML - array of tables

```
[[products]]
name = "Hammer"
sku = 738594937

points = [ { x = 1, y = 2, z = 3 },
           { x = 7, y = 8, z = 9 },
           { x = 2, y = 4, z = 8 } ]

[[products]] # empty table within the array

[[products]]
name = "Nail"
sku = 284758393

# same as:
[[points]]
x = 1
y = 2
z = 3

[[points]]
x = 7
y = 8
z = 9

[[points]]
x = 2
y = 4
z = 8
```

- S toml se klasicky pracuje pomocí knihovny v daném jazyce
- TOML se používá v cloudových službách

- YAML

- Podobný jako toml, tedy je very human readable a jednoduchý na práci s knihovnami v daných jazycích

YAML - sequence / array / list

```
# A list of tasty fruits
- Apple
- Orange
- Strawberry
- Mango
```

sequence:
dash, space, value

comment

- ['Apple', 'Orange', 'Strawberry', 'Mango']
- Jsou tam zase key-value pairs (říká se tomu tady ale mapping)
 - Name: Martin
 - Job: developer
- Velmi zásadní je indentace

YAML - mapping / map / dictionary / hash

```
# An employee record
name: Martin D'vloper
job: Developer
skill: Elite

# Employee records
- martin:
  name: Martin D'vloper
  job: Developer
  skills:
    - python
    - perl
    - pascal
- tabitha:
  name: Tabitha Bitumen
  job: Developer
  skills:
    - lisp
    - fortran
    - erlang

martin: {name: "Martin D'vloper", job: "Developer", skill: "Elite"}
```

mapping:
key, colon, space, value

indentation
mandatory and
meaningful!

- Multiline strings – dám pipe za dvojtečku a pak text (zachovám newlines), nebo mohu použít > (newlined se pak zbavím)
- Pro práci s yamlem jsou zase knihovny

YAML - complex mapping keys

? indicates complex mapping key

- sequence
- mapping

```

---
?
- "Detroit Tigers"
- "Chicago cubs"
:
- 2001-07-23
?
- "New York Yankees"
- "Atlanta Braves"
:
- 2001-07-02
- 2001-08-12
- 2001-08-14
  
```

Diagram illustrating complex mapping keys in YAML. The left side shows a sequence of keys (e.g., "Detroit Tigers", "Chicago cubs") and values (e.g., 2001-07-23). The right side shows a mapping key (e.g., "New York Yankees") and its corresponding value (e.g., 2001-07-02). Red arrows point from the keys to the values.

○

YAML - core schema

Core Schema

- extension of JSON schema, towards human readability

Regular expression	Resolved to tag
null Null NULL ~	tag:yaml.org,2002:null
/* Empty */	tag:yaml.org,2002:null
true True TRUE false False FALSE	tag:yaml.org,2002:bool
[~+]? [0-9]+	tag:yaml.org,2002:int (Base 10)
0o [0-7]+	tag:yaml.org,2002:int (Base 8)
0x [0-9a-fA-F]+	tag:yaml.org,2002:int (Base 16)
[~+]? (\. [0-9]+ [0-9]+ (\. [0-9]*)?) ([eE] [~+]? [0-9]+) ?	tag:yaml.org,2002:float (Number)
[~+]? (\.inf \.Inf \.INF)	tag:yaml.org,2002:float (Infinity)
\.nan \.NaN \.NAN	tag:yaml.org,2002:float (Not a number)
*	tag:yaml.org,2002:str (Default)

○

YAML - anchors and aliases

Diagram illustrating anchors and aliases in YAML. The left side shows the original YAML code with anchors and aliases. The right side shows the resolved YAML code where the anchors are replaced by their values.

```

---
center: &ORIGIN {x: 73, y: 129}
radius: 7

start: *ORIGIN
finish: { x: 89, y: 102 }

start: *ORIGIN
color: 0xFFEEBB
text: Pretty vector drawing.
  
```

Resolved YAML:

```

---
center:
  x: 73
  y: 129
radius: 7

finish:
  x: 89
  y: 102
start:
  x: 73
  y: 129

color: 16772795
start:
  x: 73
  y: 129
text: "Pretty vector drawing."
  
```

○

- Yaml se používá v docker compose, kubernetes config

12. Přednáška – Multimediální formáty

- Grafické formáty

- Dvě hlavní reprezentace, vektorová a rastrová (pixelová)
- Vektorová grafika
 - 2d grafika, jsou to nějaké tvary s ohraničením

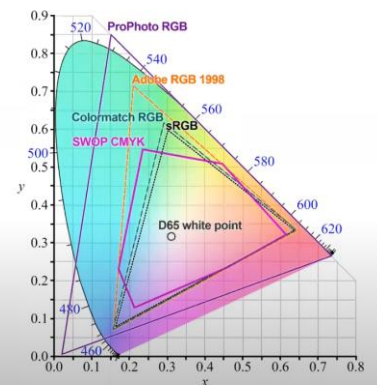
- Lze tam mít i text s fontem a barvou
- Reprezentuje se to SVG (hostitelský formát XML)
- Lze SVG embeddovat do html
- Používá se hodně na diagramy, grafy, plot charts a inženýrské návrhy/plány
- Rastrová grafika (bitmap)
 - Základní stavební kámen jsou pixely (tečky)
 - Monochromatické – černobílé
 - Grayscale – nabývá i více odstínů šedi
 - Škála barev je pochopitelně reprezentována RGB, je hlavně pro displeje
 - Kromě rgb je ještě cmyk (cyan, magenta, yellow, black), ta je hlavně pro tiskárny
 - Barevný prostor je standardizovaný, zahrnuje nějakou podmnožinu barev, kterou lze reprezentovat

Specific **color spaces**, shown as polygons on the CIE 1931 diagram

Color space has standardized hues of primaries

Color model, e.g. RGB, needs to be **mapped to** specific **color space**.

- sRGB
 - standard Red Green Blue
 - by HP & Microsoft for use on the Internet
 - endorsed by W3C, Exif, Intel, Corel, ...
- Adobe RGB
 - Designed to encompass colors achievable by CMYK printers
 - but by using RGB
- DCI P3, Rec 2020, ...



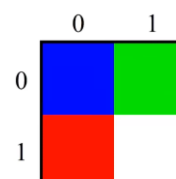
- Gamut – všechny barvy, které lze zobrazit na nějakém zařízení (pokrytí většího barevného prostoru)
- Bitová hloubka – počet barev
- RGBA – přidává alpha kanál, který reprezentuje průhlednost
- Dithering – technika pro takovou kombinaci barev, kde třeba pomocí 2 barev vytvořím další barvu (ale stále mám k dispozici jen ty 2 původní, tedy jen vlastně vizuálně)

- Třeba tedy více pixelů je použito k reprezentaci jednoho pixelu jiné barvy

- Formáty:
- DIP – device independent bitmap (bez komprese)

BMP - Device-Independent Bitmap (DIB)

Offset	Size	Hex value	Value	Description
BMP Header				
DIB Header				
Start of pixel array (bitmap data)				
36h	3	00 00 FF	0 0 255	Red, Pixel (1,0)
39h	3	FF FF FF	255 255 255	White, Pixel (1,1)
3Ch	2	00 00	0 0	Padding for 4 byte alignment (could be a value other than zero)
3Eh	3	FF 00 00	255 0 0	Blue, Pixel (0,0)
41h	3	00 FF 00	0 255 0	Green, Pixel (0,1)
44h	2	00 00	0 0	Padding for 4 byte alignment (could be a value other than zero)



- Tato reprezentace je velmi zbytková, velmi velké soubory
- Lze zavést lossless compression
 - Run length coding
 - Mohu říkat kolik políček dané barvy mám
 - Mohu indexovat barvy, tedy na jakém pixelu končí daná barva
 - Blockwise coding
 - Rozdělím si obrázek na čtvercové bloky a řeknu na jaké pozici jsou jaké pozici jsou jaké barvy, využiji takhle oba rozměry
 - Quadtree coding
 - Každý obrázek rozdělíme na čtvrtiny, pokud je jedné barvy, tak konec, pokud není, tak ho zase rozdělím na čtvrtiny
 - Další jsou ještě huffman coding, LZ77 komprese
- GIF – graphics interchange format
 - 8 bitů na pixel, každá barva je z 24-bit RGB modelu
 - Používá LZW kompresi, uměli animace
- PNG – portable network graphics
 - Úplně plný rgb model a navíc průhlednost
 - Neumí ale animace
 - Používá DEFLATE lossless kompresi
- Lossy compression – funguje pomocí diskretní kosínové transformace (detaily v přednášce)
 - Kvantizace
 - Zahodíme vysoké frekvence z obrázku
 - Vychází z toho, že lidé jsou citlivější na nižší frekvence
 - Podle množství vyhozených vysokých frekvencí mohou kontrolovat míru komprese
 - Chroma subsampling
 - Používá jiný barevný model – YCbCr (jas, červená, modrá) - příklad v přednášce, to bude přesnější :D
- JPEG – joint photographic experts group
 - Velmi starý, vznikl v 1992



- Loss vs lossless compression
 - Lossless se hodí na obrázky, kde jsou větší plochy stejné barvy – screenshoty a malůvky – PNG, WebP, AVIF
 - Lossy se používá třeba u fotografií – JPEG, WebP, AVIF
- Raw formáty
 - Ukládá se přímo to, co se dostane ze senzorů
 - Obsahuje tedy i intenzitu světla v jednotlivých pixelech
- Video formáty
 - R210
 - Velmi jednoduchý formát, skutečně jen rastrové obrázky za sebou
 - Není tam žádná komprese, je to velmi datové náročné, velké
 - MJPEG – motion jpeg
 - Video je vlastně sekvence jpeg obrázků za sebou
 - Typicky má 20:1 poměr komprese
 - Video compression – Inter-picture prediction
 - Snímek se rozdělí na makrobloky – bloky se zkomprimují
 - Pak se pro každý blok přidá informace, kam se daný blok posune v dalším snímku videa, tedy pokud tam jsou nějaké statické objekty, tak se nemusí generovat znova, což výrazně ušetří
 - H.261
 - Velmi stará a omezená, umí jen 2 rozlišení
 - MPEG
 - Používal se na video cd a pak i pro videa na internetu
 - Ve videu byly snímky různých typů
 - Byl snímek, který byl reprezentován celý a pak další byly reprezentovány jako ty makrobloky (tedy reprezentován jako rozdíl vůči původnímu obrázku)
 - MPEG-2
 - Přišel s DVD videem, HD DVD, Blu-ray
 - Používal se i na TV HDTV vysílání
 - H.263
 - Na low-bitrate vide, přišel s MMS
 - MPEG-4
 - Přineslo možnost přesunutí celého obrázku, nejen makrobloků
 - Lepší na manipulaci s jednotlivými pixely
 - Přineslo lepší kódování videa, dodnes se používá (MPEG-4 Advanced Video Coding)
 - MPEG-H
 - Přinesl ještě lepší encoding (HEVC), lepší až o 50 % než AVC
 - VP8/9 (VP9 dodnes na YouTube)
 - Otevřený video formát, na rozdíl od MPEG

- AV1
 - Zase open a je založený na VP9
- H.266
 - Úplně nový z roku 2020 a uzavřený
 - Umí 360stupňové video
- Digitální audio formáty
 - PCM (pulse-code manipulation) – digitální reprezentace zvukové vlny
 - Amplituda je měřena na intervalech a pak se zaokrouhlí na nejbližší hodnotu v nějakém digitální rangei
 - WAV
 - Formát pro nekomprimované audio
 - FLAC
 - Umí lossless compression – aproximuje vlnu pomocí nějaké funkce + se přidává chyba
 - MP3
 - Používá ztrátovou kompresi audia
 - Zase funguje pomocí diskrétní kosínové transformaci (zahazuje frekvence, kterých si člověk stejně nevšimne)
 - AAC
 - Nástupce MP3
 - Lepší komprese atd.
 - OPUS
 - Otevřený formát z roku 2019
 - Velmi efektivní v porovnání s uzavřenými
- Multimediální formáty (kontejnerové formáty)
 - Formáty s videem i audiem + titulky třeba
 - Je třeba zajistit seeking, informace o kapitolách atd.
 - Přidávají hlavně metadata
 - AVI – audio video interleave
 - Simple containers
 - JPEG, PNG, WAV
 - TIFF - images and metadata
 - Flexible containers - patented, licensed
 - AVI - Audio Video Interleave - .avi
 - MPEG program stream - .ps .mpg .mpeg
 - MPEG-2 transport stream - .ts .m2t
 - MP4 - .mp4
 - Flexible containers - open, royalty-free
 - Matroska - .mkv
 - its subset WebM - .webm
 - Ogg - .ogg
 - OGG je od tvůrců OPUS, je tedy open
 - WebM je open od Googlu

- Print formáty

- PostScript
 - Programovací jazyk pro tiskárny
 - Má to svůj media type – application/postscript
 - Dnes je to jako encapsulated postscript = postscript + ecapsulated bitmap preview
- PDF – portable document format
 - Od adobe z roku 1993
 - Dnes funguje podle specifikace vydané v roce 2020
 - Je založené na postscriptu
- PDF/A
 - Zakazuje linkování knihoven, fontů atd., vše musí být zabaleno do jednoho souboru i s tím PDF
 - Zakazuje šifrování, žádný javascript atd.

Additional PDF based standards



[PDF/X](#)

ISO 15930 2001

- reliable print data eXchange

[PDF/UA](#)

ISO 14289 2012

- Universal Accessibility
- ensures accessibility on screen readers and other assistive technologies

[PDF/VT](#)

ISO 16612-2 2010

- based on PDF/X
- Variable data and Transactional printing
- invoices, marketing documents
 - where text needs to be changed in places

[PDF/E](#)

[ISO 24517](#) 2008

- Engineering documents
- Rotating and folding 3D objects in U3D

- Pro editaci pdfek je adobe illustrator

13. přednáška – Formáty pro textové dokumenty

- RFC 1341 – Rich text

- Velmi jednoduchý syntax pro formátování textu

Richtext

[RFC 1341](#), 1992

- Extremely simple, yet extensible syntax
 - formatting commands between < and >
 - command no more than 40 characters
 - may be preceded by /, making them negations
 - balanced formatting commands and negations, with 3 exceptions
 - <lt>, <nl>, <np>
- Extremely limited capabilities
- Compatibility with SGML

US-ASCII encoding, can be explicitly switched

- Media type: text/richtext

```
<bold>Now</bold> is the time for
<italic>all</italic> good men
<smaller>(and <lt>women)</smaller> to
<ignoreme></ignoreme> come
to the aid of their
<nl>
beloved <nl><nl>country. <comment> Stupid
quote! </comment> -- the end
```

```
Now is the time for all good men (and women) to
come to the aid of their
beloved
country. -- the end
```

- Enriched text
 - o Vylepšený rich text, měl nějaké syntaktické zkratky
 - o Stal se pak standardem pro emaily
- RTF – rich text format
 - o Microsoft vydal
 - o Bylo to mířeno na plain text soubory
 - o Nebyl otevřený
- 602
 - o Československý formát
 - o Pro formátování používal znaky z ascii tabulky, které byly netisknutelné
- HTML
 - o Hlavně pro internetovou komunikaci
 - o Standard html spravovalo W3C, dnes už to je WHATWG
 - o Markup = je to text, který má vizuálně odlišitelné formátovací řetězce
- Markdown
 - o Je to markup language
 - o Aktuální specifikace je z roku 2004
 - o Html je publishing format, ale markdown je writing format (jsou na sebe snadno převoditelné)
 - o Má také navíc podporu pro kusy kódu, pomocí backticků
 - o Markdown má velké množství implementací, kde každá má svoje další formátování (dialekty)
 - o CommonMark
 - Měl by být standardizovaný
 - Fenced code block, mohu za backticky dát id programovacího jazyka a pak se to samo naformátuje, jak má
 - Do markdown dokumentu lze vždy dát nějaký html element, pokud mi nestačí markdown formátování
 - o Markdown se používá pro readme.md soubory, tedy github, gitlab
 - o Pak se používá pro staticky generované stránky, tedy stránky v markdown byly servírovány jako html
- Wikitext
 - o Nemá formální specifikaci
 - o Slouží pro formátování textu na všech wiki stránkách
- TeX
 - o Otevřený, typicky se používá pro vědecké texty, kvůli dobré manipulaci se vzorci atd.
 - o Je to systém pro tu hezkou manipulaci s textem
- LaTeX
 - o Je to vlastně knihoven maker a balíčku, které jsou napsány v TeXu (ten je moc low-level)

- Výsledek je PDF soubor

LaTeX - basics - commands, simple document

Commands start with backslashes
take parameters in `[]` and `{}`

`\documentclass{class}`

- chooses template/style of the document
 - article - no chapters
 - report - with chapters
 - ...

`\begin{environment}` and `\end{environment}`

- Marks the start and end of a certain environment, e.g.
 - document
 - definition
 - ...

`\documentclass{article}`

`\begin{document}`

First document. This is a simple example, with no extra parameters or packages included.

`\end{document}`

First document. This is a simple example, with no extra parameters or packages included.

- - Příkazy začínají backslashem

LaTeX - basics - preamble, packages, math mode

LaTeX packages provide functionalities

[CTAN: Comprehensive TeX Archive Network](http://www.ctan.org/)

- Currently 6000+ packages

Packages are defined in **preamble**

- using `\usepackage[options]{package}`
- after `\documentclass{}`
- before `\begin{document}`

`\documentclass{article}`

`\usepackage{amsmath}`

`\begin{document}`

Hello world!

`\[`

`\binom{n}{k} = \frac{n!}{k!(n-k)!}`

`\]`

`\end{document}`

- E.g. package `amsmath` provides the `\binom` and `\frac` commands
- Lze specifikovat i encoding (default je utf-8)
- Lze používat section, subsection atd., i se to automaticky čísluje
- Odstavce se dělají 2 new lines
- Jsou tam i seznamy, číslované i nečíslované
- Jsou tam i odkazy na jiné sekce v textu, je to pomocí příkazu `label` nebo `autoref` (je to automatictější)
- Jsou tam pak i obrázky, je na to prostředí `figure` (commandy na centrování, `caption`, `lable` atd.)
- Jsou tam i tables (definují oddělovače, header atd.), matematika, vzorce
- Pro reference a citace je subsystém BibTeX
- V textu lze hledat exact match, wildcards nebo regulární výrazy
- Lze dokument i filtrovat podle toho co obsahují – indexace
- Lze dělat i diff a text comparison
- Named entity recognition
 - Hledáme v textu pojmenované entity – Dbpedia Spotlight