MDF - cheatsheet

- Tento cheatsheet jsem sestavil podle mých zkušeností, co se mi zdálo náročnější, nebylo řádně vysvětleno.
- Uvádím, jak jsem uvedené věci dělal já nebo jak jsem je pochopil já, určitě jdou dělat i jinými způsoby, které jsou také validní.
- Viděl jsem že po prvním běhu předmětu již p. Klímek udělal nějaké změny v zadání jednotlivých částí, tak něco nemusí sedět.
- Nezaručuju naprostou správnost uvedených věcí, prošlo to kontrolou u p. Klímka, pokud by se zde objevila nějaká chybka, tak mi dejte vědět do diskuse, díky :))
- Uvedené příklady jsou vymyšlené, nejsou z mé semestrální práce

1] Konceptuální schéma

- za mě je asi nejlepší https://www.diagrams.net/, doporučuju si nechat po celou dobu uložený nejenom .png/.svg soubor, ale i .drawio (může se stát, že v průběhu semestrálky budete muset změnit nějaké věci v konceptuálním schématu, tak si to z tohoto souboru lehce načtete a nemusíte to "kreslit" znovu (já jsem po zpětné vazbě po prvním checkpointu musel přidávat ještě jednu entitu kvůli lepší reprezentaci)
- povinnou účast ve vazbě dávejte na co nejméně míst, opravdu tam, kde VŽDY musí být nějaká data - trochu to pak ulehčí práci při validaci dat později
- popis diagramu v domain.txt veďte strukturovaně
    - Název modelu
    - Krátký popis, co modeluje, jaká tam budou data
    - Popis entit
        - Název entity
        - Slovní popis entity a vazeb se "sousedními" entitami
        - Výčet atributů
            - název
            - datový typ (číslo, datum, text atd.)
            - nějaké omezení (např. věk bude určitě kladný)

2] RDF

- háčky a čárky v IRI nevadí, u URI by vadily (jsou nahrazeny %-notací)
- rychlý výčet prefixů, které jsem používal:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/vocabulary/> .
```

- potom na custom prefixy (vymyšlené - nemusíme používat aktuální slovníky) jsem používal tento syntax:

```
@prefix cur: <http://example.org/<nazev>/> .
```

    - kde <nazev> je placeholder pro “téma” slovníku, pro auta třeba cars, pro zaměstnance staff atd.
- já jsem začal definicí jednotlivých entit, pak vazeb mezi nimi a pak propojení těma vazbama, nahodím příklad:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex: <http://example.org/vocabulary/> .

# custom prefix
@prefix car: <http://example.org/cars/> .
@prefix dri: <http://example.org/drivers/> .


# set-up první třídy
ex:Car a rdfs:Class ;  # absolutní IRI ex:Car je
http://example.org/vocabulary/Car
    rdfs:label "Auto"@cs . # nezapomínat na language tagy!

# --- definice atributů ---
ex:numOfWheels a rdf:Property ;
    rdfs:label "Počet kol"@cs ;
    rdfs:domain ex:Car ;  # entita, ke které atribut “připojujeme”
    rdfs:range xsd:integer .  # “obor hodnot”

ex:brand a rdf:Property ;
    rdfs:label "Značka auta"@cs ;
    rdfs:domain ex:Car ;
    rdfs:range rdf:langString .
# --- konec definice atributů ---

# naplnění daty:

# --- jedna instance třídy Car ---
car:Škoda a ex:Car ;  # absolutní IRI car:Škoda je
http://example.org/cars/Škoda
    ex:numOfWheels 4 ;
    ex:brand "Škoda Octavia"@cs .

# --- další instance třídy Car ---
car:Peel a ex:Car ;
    ex:numOfWheels 3 ;
    ex:brand "1962 Peel P50"@cs .

# ----------------------------------------------

# set-up druhé třídy
```

```
ex:Driver a rdfs:Class ;
    rdfs:label "Řidič"@cs .

# --- definice atributů ---
ex:firstName a rdf:Property ;
    rdfs:label "Křestní jméno"@cs ;
    rdfs:domain ex:Driver ;
    rdfs:range rdf:langString .

ex:age a rdf:Property ;
    rdfs:label "Věk"@cs ;
    rdfs:domain ex:Driver ;
    rdfs:range xsd:integer .

ex:description a rdf:Property ;
    rdfs:label "Popis vzhledu"@cs ;
    rdfs:domain ex:Driver ;
    rdfs:range xsd:integer .

# --- jedna instance třídy Driver —
dri:Pepik123 a ex:Driver ;
    ex:firstName "Josef"@cs ;
    ex:age 45 ;
    ex:description "A very tall man with a beard and strong
glasses. Always wears checked shirts and jeans."@en,
     "Vysoký muž s plnovousem a silnými brýlemi. Vždy nosí
kostkované košile a džíny"@cs . # příklad multiplikativního
atributu, který je požadován v konceptuálním schématu

# ----------------------------------------------
# definice vazby mezi Car a Driver

ex:drives a rdf:Property ; # vazba se bere jako další "atribut"
entity
    rdfs:label "Řídí"@cs ;
    rdfs:domain ex:Driver ; # entita "od které jde vazba"
    rdfs:range ex:Car .

# definice vztahu mezi Car a Driver pomocí definované vazby

dri:Pepa123 ex:drives car:Škoda .
dri:Pepa123 ex:drives car:Peel .
# zde řidič s IRI Pepa123 řídí obě auta
```

3] SPARQL

- řídí se velmi podobným syntaxem jako SQLko, myslím, že je to na slidech přednášky/cvičení celkem dobře vysvětlené

4] XML

- tady je akorát trochu oříšek v tom, jak správně reprezentovat N:M vazby
- je několik způsobů, já zkusím jeden z nich popsat
    - mám jeden hlavní .xml soubor, kde pod společný kořenem mám všechny entity, něco v tomto stylu:

```
<root xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="schema-root.xsd">

<cars>
     <car>...</car>
     <car>...</car>
</cars>
<drivers>
     <driver>...</driver>
</drivers>

</root>
```

- jak je vidno, tak je includnuté jedno schéma pro celý root, ve kterém mám na includované schémata všech entit, každá entita má svoje schéma ve vlastním souboru
- dávám příklad i s validní hlavičkou:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="https://www.w3.org/2009/01/xml.xsd" />

<xs:include schemaLocation="schema-1-cars.xsd"></xs:include>
<xs:include schemaLocation="schema-2-drivers.xsd"></xs:include>

<xs:element name="root">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="car"></xs:element>
      <xs:element ref="driver"></xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

```
</xs:schema>
```

- povinnost ve vazbě a počet instancí se ve schématu řeší přes minOccurs a maxOccurs atributy
    - nějaký element není povinný? minOccurs=”0”
- N:M vazby jsem řešil odkazováním se přes IRI, tedy každá instance v sobě měla IRI instancí jiných tříd, které k ní byly navázané, zkusím načrtnout:

```
<car>
      <iri>http://example.org/cars/Škoda</iri>
      <brand xml:lang=”cs”>Škoda</brand>
      <numOfWheels>4</numOfWheels>
      <drivers>

<driverIri>http://example.org/drivers/Pepa123</driverIri>
      </drivers>
</car>
```

- z pohledu řidiče:

```
<driver>
      <iri>http://example.org/drivers/Pepa123</iri>
      <firstName xml:lang=”cs”>Josef</firstName>
      <age>4</age>
      <description xml:lang=”cs”>Vysoký muž s…</description>
      <description xml:lang=”en”>A very tall man…</description>
      <cars>
            <carIri>http://example.org/cars/Škoda</carIri>
            <carIri>http://example.org/cars/Peel</carIri>
      </cars>
</driver>
```

- u řidiče je zase vidět řešení multiplicity atributu
- ještě je potřeba si definovat svůj custom datový typ, kde vynucujeme použití atributu xml:lang (kvůli language tagům a aby nebyla ztracena informace mezi různými reprezentacemi dat
    - dá se pro něj udělat vlastní schéma a pak ho jen importovat do ostatních schémat, kde už jenom u elementů přidáme atribut type=”stringWithLangTag”
        - název je pro ukázkové účely dlouhý, není problém si ho zkrátit :)

```
<xs:complexType name="stringWithLangTag">
  <xs:simpleContent>
      <xs:extension base="xs:string">
          <xs:attribute ref="xml:lang" use="required"/>
      </xs:extension>
  </xs:simpleContent>
</xs:complexType>
```

- v XPathu se normálně mohu dotazovat na IRI a podle toho selectovat konkrétní data pro dotaz
- v XSLT se v loopu lze spojit přes IRI takto (uložím si aktuální IRI do proměnné a tu pak používám v XPath selectech: (not sure, jestli je to nejlepší způsob, ale funguje)

```
<xsl:for-each select="cars/driver_iri">
     <xsl:variable name="iri_driver"><xsl:value-of
select="."></xsl:value-of></xsl:variable>
     <div class=”drivers”>
          <h2><xsl:value-of select=”//drivers/driver/[iri =
$iri_driver]/firstName”></h2>
     </div>
</xsl:for-each>
```

5] JSON

- zde je možné každou entitu mít v separátním souboru se svým schematem, který ho validuje
- jinak strukturou je to dost podobné jako XML
- lze postupovat tak, že jako root dám array objektů a každý objekt představuje jednu instanci té konkrétní třídy (1 třída = 1 soubor, pokud jsou mezi nimi N:M či nějaké složitější vazby, pokud je mezi nimi vazby 1:1, lze je podobně jako v XML “vnořit” do sebe a není potřeba pro ně dělat separátní soubor
- v objektu napište klasicky data instance v klasické JSON formátu
    - já jsem si navíc přidal properties “iri” a “typ”, ale není to potřeba, pokud to obejdete v @context
- zkusím zase načrtnout formát, jak by to mohlo vypadat:

```
[
{
    "@context": {
      "ex": "http://example.org/vocabulary/",
      "dri": "http://example.org/drivers/",
      "car": "http://example.org/cars/",
      "iri": "@id", → toto znamená, že id této instance je v
property “iri”
      "typ": "@type", → to stejné jako o řádek výše
      "firstName": {
        "@id": "ex:firstName",
        "@container": "@language"
      },
      "age": {
        "@id": "ex:age"
      },
      "cars": {
        "@id": "ex:drives",
```

```
        "@type": "@id"
      }
    },
    "iri": "dri:Pepa123",
    "typ": "ex:Driver",
    "firstName": {
      "cs": "Josef"
    },
    "age": 45,
    "cars": [
      "car:Škoda",
      "car:Peel"
    ]
}
]
```

- schématem validujeme jenom JSON část (ne context) a to se dá pěkně okoukat z slidů u cvičení
- .jq dotazy mohou být vcelku jednoduché

6] CSV

- zde vám moc nepomohu, tuto část jsem celou nestihl do termínu :) a nejsem si příliš jistý její správností a nerad bych zde psal věci, které nemám ověřené
- jenom ve slidech se to nikde nepíše, pro hezký output formát, který pak vypadá jako naše RDFko se hodí přepínač `--output-format turtle`

Závěr

- snad vám to pomohlo :)
- kdybyste našli chybu napište na FW do diskuse, budu tam koukat a opravím to
- nebojte se psát p. Klímkovi na Teams nebo si na něj připravit nějaké otázky před/po cvičení
- uvedené příklady jsou vymyšlené, nejsou z mé semestrální práce
- můj tip: fakt zkuste udělat tu část semestrálky co nejdřív po cvičení z té části, nezabere to tolik času, ještě to budete mít v paměti a pak to nemusíte dohánět na poslední chvíli ve zkouškovým, kdy se musíte soustředit na (pravděpodobně) důležitější předměty