Data Cube

NDBI046: Practical class 7

Prerequisite: Setting up Python (Linux, macOS)

- *Check* which *version of Python* is installed (if any)
  - If Python 3 is not installed, download the (latest) version of Python[#1] and follow the installation
- Once installed, *create* any *folder for your NDBI046 project* and navigate to it, e.g., ~/Projects/python-ndbi046
- *Create* your *Python environment*, e.g., ndbi046_env
- *Activate* you Python environment
- *Install* the required *packages*
  - Download the requirements.txt file from the practical class website
  - You may also *install additional packages*
  - You may always export the list of installed packages to a file
- Exit the Python environment after completing the practical class (not before)
  - You can return to the environment at any time by activating it

```
1  % python3 --version
2
3  % mkdir ~/Projects/python-ndbi046
4  % cd ~/Projects/python-ndbi046
5
6  % python3 -m venv ndbi046_env
7
8  % source ndbi046_env/bin/activate
9
10 (ndbi046_env) % pip install -r requirements.txt
11
12 (ndbi046_env) % pip install pandas
13
14 (ndbi046_env) % pip freeze > requirements.txt
15
16 (ndbi046_env) % deactivate
17
18 % cat requirements.txt
```

checking the installed version

creating and activating a virtual environment

installation of the required packages

export of installed packages

list installed packages

deactivating a virtual environment

#1 https://www.python.org/downloads/

Prerequisite: Setting up Python (Windows)

- *Check* which *version of Python* is installed (if any)
  - If Python 3 is not installed, download the (latest) version of Python[#1] and follow the installation
- Once installed, *create* any *folder for your NDBI046 project* and navigate to it, e.g., C:\Projects\python-ndbi046
- *Create* your *Python environment*, e.g., ndbi046_env
- *Activate* you Python environment
- *Install* the required *packages*
  - Download the requirements.txt file from the practical class website
  - You may also *install additional packages*
  - You may always export the list of installed packages to a file
- Exit the Python environment after completing the practical class (not before)
  - You can return to the environment at any time by activating it

checking the installed version

```
> python3 --version

> mkdir C:\Projects\python-ndbi046
> cd C:\Projects\python-ndbi046

> python3 -m venv ndbi046_env

> ndbi046_env\Scripts\activate.bat

(ndbi046_env) > pip install -r requirements.txt

(ndbi046_env) > pip install pandas

(ndbi046_env) > pip freeze > requirements.txt

(ndbi046_env) > deactivate

> type requirements.txt
```

creating and activating a virtual environment

installation of the required packages

export of installed packages

list installed packages

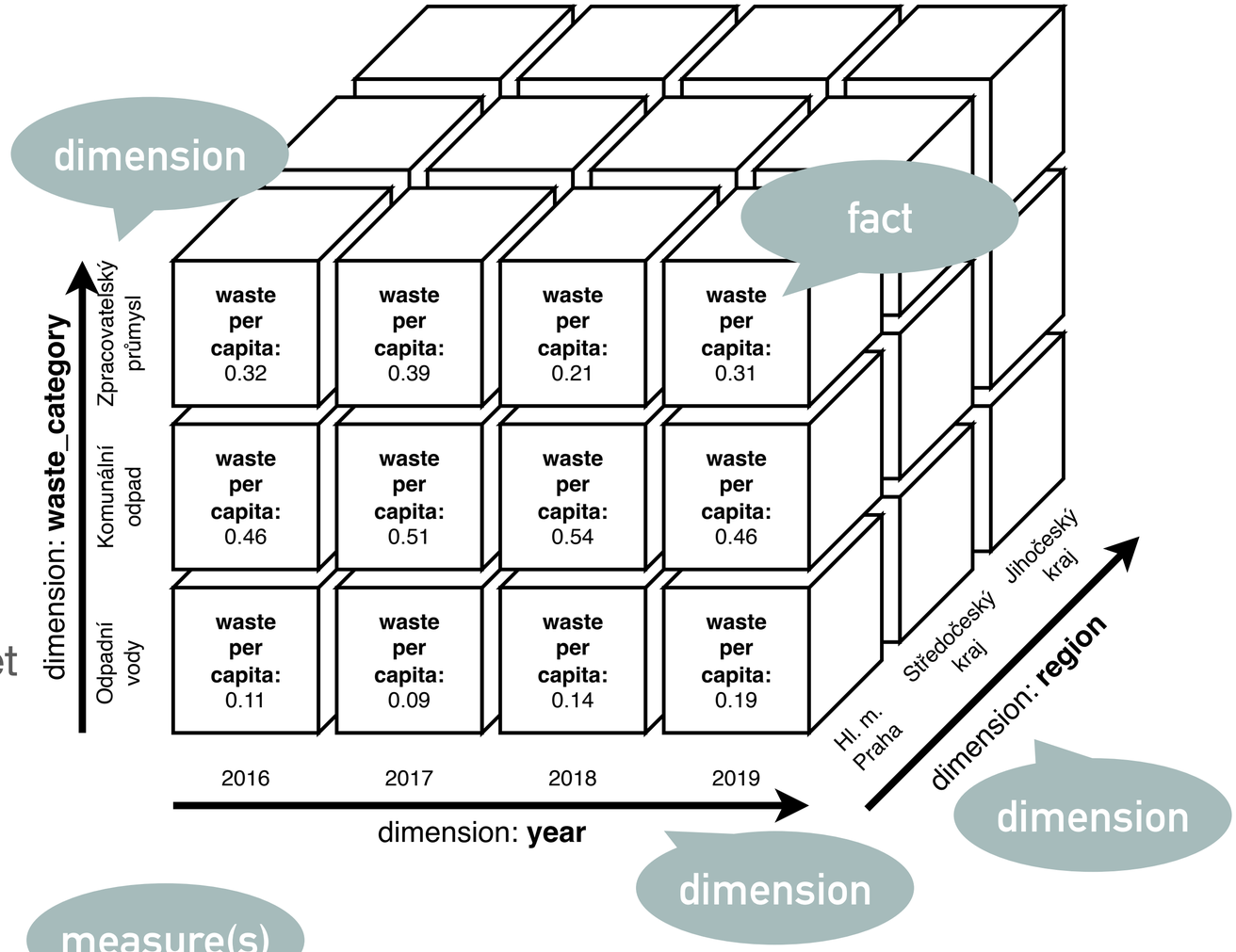
deactivating a virtual environment

#1 https://www.python.org/downloads/

Data Cube

- A *multi-dimensional dataset* that provides a structured way to aggregate and summarize data
  - Enables effective analysis and reporting

- *Dimensions*
  - Identify various aspects of the data and provide a basis for categorizing data, e.g., time, geography, product categories
- *Facts*
  - The numerical value or observation(s) for a given set of dimensions
  - There can be no fact for a given set of dimensions
- *Measures*
  - Computed value along selected dimensions (e.g., aggregation functions)

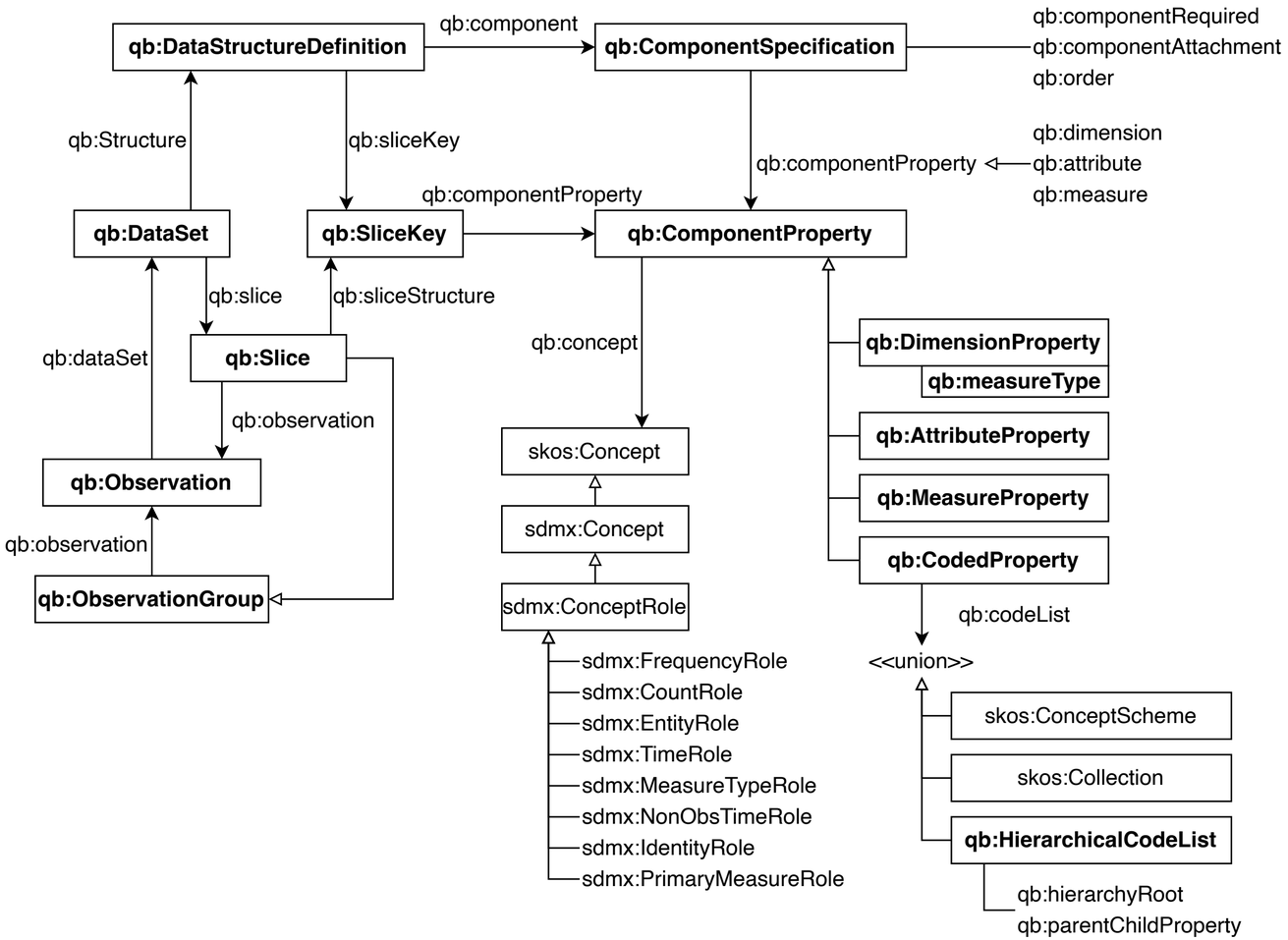


measure: **total waste production in 2018 = 6.94**

measure: **total waste production in 2019 = 7.12**

Data Cube Vocabulary

- RDF vocabulary
- W3C recommendation from January 2014
- Based on Statistical Data and Metadata eXchange (SDMX)



Source: https://www.w3.org/TR/vocab-data-cube/#outline

Data Cube: Structure

- The Data Cube vocabulary *represents* the data structure definition *as* an *RDF properties*
  - qb:DataStructureDefinition *defines the structure* of one or more datasets
  - Dimensions, attributes and measures are represented as instances of the qb:ComponentProperty subclass:
    - qb:DimensionProperty
    - qb:AttributeProperty
    - qb:MeasureProperty

- Each qb:ComponentProperty subclass can be associated with other resources, e.g.:
  - rdfs:label assigns a *human-readable label* to a resource
  - rdfs:range specifies the *class of values* that the subject property represents
  - rdfs:domain specifies the *resource class* to which the subject property can be applied

```
@prefix ndbi: <https://ndbi046.pavel.koupil/ontology#> .
@prefix qb: <http://purl.org/linked-data/cube#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

ndbi:structure a qb:DataStructureDefinition ;
  qb:component [
      qb:dimension ndbi:year ;
      qb:order 1
    ],
    [ qb:measure ndbi:waste_per_capita ],
    [ qb:dimension ndbi:waste_category ;
      qb:order 2 ],
    [ qb:dimension ndbi:region ;
      qb:order 3 ] .

ndbi:region a qb:DimensionProperty, rdfs:Property ;
  rdfs:label "Kraj"@cs, "Region"@en ;
  rdfs:range xsd:string .

ndbi:waste_category a qb:DimensionProperty, rdfs:Property ;
  rdfs:label "Kategorie odpadu"@cs, "Waste Category"@en ;
  rdfs:range xsd:string .

ndbi:waste_per_capita a qb:MeasureProperty ;
  rdfs:label "Odpad na obyvatele"@cs, "Waste Per Capita"@en ;
  rdfs:range xsd:float .

ndbi:year a qb:DimensionProperty, rdfs:Property ;
  rdfs:label "Rok"@cs, "Year"@en ;
  rdfs:range xsd:int .
```

data structure definition

dimension

measure

Example 7.1: Data structure definition

- Create a Python script that *produces* the following data *cube structure definition*:
  - *Dimensions*: year, waste_category, region
  - *Measure*: waste_per_capita
- Use rdfs:range that match the attribute types in the fact_waste table in the ndbi046 database on the webik.ms.mff.cuni.cz server

Tip: To create an RDF representation, use the rdflib library

- (see https://rdflib.readthedocs.io/en/stable/)

Example 7.1: Data structure definition (Solution)

```python
import pandas as pd
from rdflib import Graph, BNode, Literal, Namespace
from rdflib.namespace import RDF, QB, XSD, SKOS, DCTERMS, OWL
import logging
import sys

logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

NS = Namespace("https://ndbi046.pavel.koupil/ontology#")
NSR = Namespace("https://ndbi046.pavel.koupil/resources/")
RDFS = Namespace("http://www.w3.org/2000/01/rdf-schema#")

def as_data_cube(data: pd.DataFrame) -> Graph:
  pass

def create_dimensions(collector: Graph) -> list:
  pass

def create_measures(collector: Graph) -> list:
  pass

def create_structure(collector: Graph, dimensions: list, measures: list) -> BNode:
  pass
```

import of necessary libraries

definition of namespaces

decomposition of the assignment into individual tasks

Example 7.1: Data structure definition (Solution)

```python
def as_data_cube(data: pd.DataFrame) -> Graph:
  result = Graph()
  result.bind("ndbi", NS)
  result.bind("ndbi-r", NSR)
  result.bind("rdfs", RDFS)
  result.bind("rdf", RDF)
  result.bind("xsd", XSD)
  result.bind("owl", OWL)

  dimensions = create_dimensions(result)
  measures = create_measures(result)
  structure = create_structure(result, dimensions, measures)
  return result
```

bind namespaces with prefixes

```python
def create_measures(collector: Graph) -> list:
  waste_per_capita = NS.waste_per_capita
  collector.add((waste_per_capita, RDF.type, QB.MeasureProperty))
  collector.add((waste_per_capita, RDFS.label, Literal("Waste Per Capita", lang="en")))
  collector.add((waste_per_capita, RDFS.range, XSD.float))

  return [waste_per_capita]
```

specify resource type

specify a resource label

specify a resource range

Example 7.1: Data structure definition (Solution)

```
1  def create_dimensions(collector: Graph) -> list:
2    year = NS.year
3    collector.add((year, RDF.type, RDFS.Property))
4    collector.add((year, RDF.type, QB.DimensionProperty))
5    collector.add((year, RDFS.label, Literal("Rok", lang="cs")))
6    collector.add((year, RDFS.label, Literal("Year", lang="en")))
7    collector.add((year, RDFS.range, XSD.int))
8
9    waste_category = NS.waste_category
10   collector.add((waste_category, RDF.type, RDFS.Property))
11   collector.add((waste_category, RDF.type, QB.DimensionProperty))
12   collector.add((waste_category, RDFS.label, Literal("Kategorie odpadu", lang="cs")))
13   collector.add((waste_category, RDFS.label, Literal("Waste Category", lang="en")))
14   collector.add((waste_category, RDFS.range, XSD.string))
15
16   region = NS.region
17   collector.add((region, RDF.type, RDFS.Property))
18   collector.add((region, RDF.type, QB.DimensionProperty))
19   collector.add((region, RDFS.label, Literal("Kraj", lang="cs")))
20   collector.add((region, RDFS.label, Literal("Region", lang="en")))
21   collector.add((region, RDFS.range, XSD.string))
22
23   return [year, waste_category, region]
```

assign the resource represented by the namespace prefix NS

specify the resource type

specify the property to be related to the dimension in the RDF data cube

Example 7.1: Data structure definition (Solution)

initializes Data Structure Definition resource

```
1  def create_structure(collector: Graph, dimensions: list, measures: list) -> BNode:
2    structure = NS.structure
3    collector.add((structure, RDF.type, QB.DataStructureDefinition))
4
5    for index, dimension in enumerate(dimensions):
6      component = BNode()
7      collector.add((structure, QB.component, component))
8      collector.add((component, QB.dimension, dimension))
9      collector.add((component, QB.order, Literal(index + 1)))
10
11   for measure in measures:
12     component = BNode()
13     collector.add((structure, QB.component, component))
14     collector.add((component, QB.measure, measure))
15
16   return structure
```

creates a blank node to represent a component of the structure

adds a triples indicating that the component is part of the structure, is a dimension, and indicates the order of the dimension

adds the measure as part of the structure

Data Cube: Dataset

- A dataset is a *collection of statistical data* that conforms to a defined structure
  - The resource representing the dataset is typed as qb:DataSet and linked to the *corresponding data structure definition* via the qb:structure property
  - qb:Observation represents *single observation*
    - Identified by a unique resource URI
    - The values for *each of the attributes*, *dimensions*, and *measurements are attached*
    - Linked to the containing data set using the qb:dataSet property

```
ndbi-r:dataCubeInstance a qb:DataSet ;
  rdfs:label "Waste Data Cube"@en ;
  qb:structure ndbi:structure .
```

```
ndbi-r:observation-000 a qb:Observation ;
  qb:dataSet ndbi-r:dataCubeInstance ;
  ndbi:region "Plzeňský kraj"^^xsd:string ;
  ndbi:waste_category "Celková produkce komunálního odpadu"^^xsd:string ;
  ndbi:waste_per_capita "0.25170690234898335"^^xsd:float ;
  ndbi:year "2014"^^xsd:int .
```

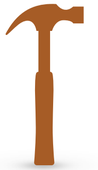
Example 7.2: Dataset

- Create a Python script that performs a *transformation* of selected columns from the fact_waste table from the ndbi046 database on the webik.ms.mff.cuni.cz server *into a data cube* represented by the *RDF Data Vocabulary*
  - In particular, extend the solution of Example 7.1:
    - Provide a dataset definition (qb:dataSet)
    - Add all observations (qb:Observation), each corresponding to one row of the fact table

Tip: To create an RDF representation, use the rdflib library

- (see https://rdflib.readthedocs.io/en/stable/intro_to_sparql.html)

Example 7.2: Dataset (Solution)



```python
def as_data_cube(data: pd.DataFrame) -> Graph:
  result = Graph()
  result.bind("ndbi", NS)
  result.bind("ndbi-r", NSR)
  result.bind("rdfs", RDFS)
  result.bind("rdf", RDF)
  result.bind("xsd", XSD)
  result.bind("owl", OWL)

  dimensions = create_dimensions(result)
  measures = create_measures(result)
  structure = create_structure(result, dimensions, measures)

  dataset = create_dataset(result, structure)
  create_observations(result, dataset, data)
```

reuse the previous solution

```python
def create_dataset(collector: Graph, structure: BNode) -> BNode:
  dataset = NSR.dataCubeInstance
  collector.add((dataset, RDF.type, QB.DataSet))
  collector.add((dataset, RDFS.label, Literal("Waste Data Cube", lang="en")))
  collector.add((dataset, QB.structure, structure))
  return dataset
```

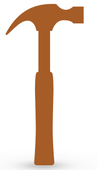
adds a triple stating that dataset is of type Qb.DataSet

specify the label od the dataset

initializes an instance of dataset

links dataset to the provided data structure definition

Example 7.2: Dataset (Solution)



```python
def create_observations(col: Graph, dataset: BNode, data: pd.DataFrame) -> None:
  for index, row in data.iterrows():
    res = NSR["observation-" + str(index).zfill(3)]

    col.add((res, RDF.type, QB.Observation))
    col.add((res, QB.dataSet, dataset))

    # Dimension: Year
    col.add((res, NS.year, Literal(row['year'], datatype=XSD.int)))

    # Dimension: Waste Category
    if pd.isna(row['cznace']) or row['cznace'] == "":
      col.add((res, NS.waste_category, Literal(row['stapro'], datatype=XSD.string)))
    else:
      col.add((res, NS.waste_category, Literal(row['cznace'], datatype=XSD.string)))

    # Dimension: Region
    col.add((res, NS.region, Literal(row['region_name'], datatype=XSD.string)))

    # Measure Waste Per Capita
    col.add((res, NS.waste_per_capita, Literal(row['waste_per_capita'], datatype=XSD.float)))
```

generate a unique resource URI for the observation

the observation is associated with a dataset

all dimensions and measures are present in the observation

Exercise 7.3: Handling multiple measures

- Create a Python script that performs a *complete transformation* of the fact_waste table from the ndbi046 database on the webik.ms.mff.cuni.cz server *into a data cube* represented by the RDF Data Vocabulary
  - Extend the data cube such that its definition and dataset contain all measures (i.e., including population and waste_amount)

- Use one of the following approaches:
  - Each observation records *one observed value for one measure*
    - That is, introduce an additional dimension whose value indicates the measure that each observation represents
  - Or *one observation* can provide values for *several different measures*

- Documentation: https://www.w3.org/TR/vocab-data-cube/#dsd-mm

DataSet metadata

- qb:DataSet should be tagged with *metadata* to *support discovery*, *presentation* and *processing*
- The recommend core set of metadata terms is:
  - dcterms:title
  - rdfs:label (may be same as dct:title)
  - dcterms:description
  - rdfs:comment (may be same as dct:description)
  - dcterms:issued
  - dcterms:modified
  - dcterms:subject
  - dcterms:publisher
  - dcterms:license

```
ndbi-r:dataCubeInstance a qb:DataSet ;
  rdfs:label "Produkce odpadů v krajích České republiky"@cs, "Waste Data Cube"@en ;
  dcterms:issued "2024-04-04"^^xsd:date ;
  dcterms:license "https://gitlab.mff.cuni.cz/contosp/ndbi046/-/blob/master/LICENSE"^^xsd:anyURI ;
  dcterms:modified "2024-04-04"^^xsd:date ;
  dcterms:publisher "Pavel Koupil"@en ;
  dcterms:title "Produkce odpadů v krajích České republiky"@cs, "Waste Data Cube"@en ;
  qb:structure ndbi:structure .
```

Source: https://www.w3.org/TR/vocab-data-cube/#metadata

Exercise 7.4: Additional dataset metadata

- Extend the solution to Example 7.2 and *provide the recommended metadata* for qb:DataSet

Concept schemes and code lists

- Dimension values within a dataset must be *unambiguously defined*
  - As *typed values* (e.g., xsd:dateTime) *or coded values* from a list of controlled terms (represented by URI references)
- SKOS represents individual code values using skos:Concept and the set of allowed values using skos:ConceptScheme or skos:Collection
  - skos:prefLabel is used to name the code
  - skos:note provides a description
- The data cube vocabulary provides a qb:concept property that associates qb:ComponentProperty with the concept it represents
  - A component can also optionally be annotated with a qb:codeList property that identifies a set of skos:Concepts used as codes (i.e., code list)
  - The value of qb:codeList may be skos:ConceptScheme, skos:Collection, or qb:HierarchicalCodeList
  - The rdfs:range of the component can be left as simply skos:Concept
    - It is recommended to define an rdfs:Class whose members are all skos:Concepts within a particular schema

concept scheme

```
sdmx-code:region a skos:ConceptScheme ;
  rdfs:label "Kraj"@cs, "Region"@en ;
  rdfs:seeAlso ndbi-r:Region ;
  skos:note
    "This code list provides a list of regions
     in Czechia."@en ;
  skos:prefLabel "Kraj"@cs, "Region"@en .
```

concept class

```
sdmx-code:Region a rdfs:Class, owl:Class ;
  rdfs:label "Kraj"@cs, "Region"@en ;
  rdfs:seeAlso ndbi-r:region ;
  rdfs:subClassOf skos:Concept ;
  skos:prefLabel "Kraj"@cs, "Region"@en .
```

concepts

```
<https://ndbi046.pavel.koupil/resources/region/1>
    a sdmx-code:Region, skos:Concept ;
  skos:inScheme sdmx-code:region ;
  skos:prefLabel "Hlavní město Praha"@cs
  skos:topConceptOf sdmx-code:region .
```

```
<https://ndbi046.pavel.koupil/resources/region/2>
    a sdmx-code:Region, skos:Concept ;
  skos:inScheme sdmx-code:region ;
  skos:prefLabel "Středočeský kraj"@cs ;
  skos:topConceptOf sdmx-code:region .
```

Source: https://www.w3.org/TR/vocab-data-cube/#schemes

SDMX

- The *SDMX* standard defines a *set of common statistical concepts* and associated *code lists* to *improve reusability* and interoperability between datasets:
  - *sdmx-concept*: http://purl.org/linked-data/sdmx/2009/concept#
    - SKOS concepts for each concept defined by COG
  - *sdmx-code*: http://purl.org/linked-data/sdmx/2009/code#
    - SKOS concepts and ConceptSchemes for each code list defined by COG
  - *sdmx-dimension*: http://purl.org/linked-data/sdmx/2009/dimension#
    - Component properties corresponding to each COG concept that can be used as a dimension
  - *sdmx-attribute*: http://purl.org/linked-data/sdmx/2009/attribute#
    - Properties of component corresponding to each COG concept that can be used as an attribute
  - *sdmx-measure*: http://purl.org/linked-data/sdmx/2009/measure#
    - Properties of the component corresponding to each COG concept that can be used as a measure

Source: https://www.w3.org/TR/vocab-data-cube/#schemes

Exercise 7.5: Concept schemes and code lists

- Introduce (*reuse*) *concepts* for region and waste_category dimensions in the data cube from Exercise 7.4
  - As with region, code the region_name values
  - As with waste_category, encode the values of not empty cznace
    - If cznace value represents an empty string, use the corresponding stapro value
  - A common convention is that the class name is the same as the concept schema name, but capitalized

- Introduced *concepts must have a label* (skos:prefLabel)
- Use *SDMX concepts* to enhance interoperability

Exercise 7.6: Slices

- Follow the Data Cube Vocabulary documentation and extend the solution from Exercise 7.5 to *include* qb:Slice *over a selected subset of observations*
  - First, *define the slice*(s) *structure* by assigning a qb:sliceKey with the data structure definition
  - Slices are represented by qb:Slice instances that reference the observation of the slice using qb:observation and the key using qb:sliceStructure

- Please refer to the documentation for further instructions:
  - https://www.w3.org/TR/vocab-data-cube/#slices

Well-formed data cube

- A *well-formed* RDF *data cube* is an RDF graph specifying one or more instances of qb:DataSet for which each of the integrity constraint check is satisfied, e.g.:
  - IC-1 *Unique DataSet*
    - Every qb:Observation has exactly one related qb:DataSet
  - IC-2 *Unique DataStructureDefinition* (DSD)
    - Every qb:DataSet has exactly one associated qb:DataStructureDefinition
  - IC-3 *DSD includes measure*
    - Every qb:DataStructureDefinition must include at least one declared measure
  - IC-4 *Dimensions have range*
    - Every dimension declared in a qb:DataStructureDefinition must have a declared rdfs:range
  - IC-5 *Concept dimensions have code lists*
    - Every dimension with range skos:Concept must have a qb:codeList

```
ndbi-r:dataCubeInstance a qb:DataSet ;
  rdfs:label
    "Produkce odpadů v krajích České republiky"@cs,
    "Waste Data Cube"@en ;
  dcterms:issued "2024-04-04"^^xsd:date ;
  dcterms:license
    "https://gitlab.mff.cuni.cz/contosp/ndbi046/
     -/blob/master/LICENSE"^^xsd:anyURI ;
  dcterms:modified "2024-04-04"^^xsd:date ;
  dcterms:publisher "Pavel Koupil"@en ;
  dcterms:title
    "Produkce odpadů v krajích České republiky"@cs,
    "Waste Data Cube"@en ;
  qb:structure ndbi:structure .
```

IC-2

```
ndbi:year a qb:DimensionProperty, rdfs:Property ;
  rdfs:label "Rok"@cs, "Year"@en ;
  qb:concept sdmx-concept:freq ;
  rdfs:range xsd:int ;
  rdfs:subPropertyOf sdmx-concept:Freq .
```

IC-4

```
ndbi:waste_category a qb:DimensionProperty, rdfs:Property ;
  rdfs:label "Kategorie odpadu"@cs, "Waste Category"@en ;
  qb:codeList sdmx-code:wasteCategory ;
  rdfs:range sdmx-code:WasteCategory ;
  rdfs:subPropertyOf sdmx-code:WasteCategory .
```

IC-4

IC-5

```
ndbi-r:observation-000 a qb:Observation ;
  qb:dataSet ndbi-r:dataCubeInstance ;
  ndbi:region
    <https://ndbi046.pavel.koupil/resources/region/4> ;
  ndbi:waste_category
    <https://ndbi046.pavel.koupil/resources/wasteCategory/1> ;
  ndbi:waste_per_capita "0.2517069023489833"^^xsd:float ;
  ndbi:year <https://ndbi046.pavel.koupil/resources/year/2014> .
```

IC-1

For complete list of integrity constraints, see documentation: https://www.w3.org/TR/vocab-data-cube/#wf-rules

Exercise 7.7: Well-formed data cube

- Write a Python *script* that *verifies* that the data cube is *well formed*
  - Validate data cubes from Exercises 7.5 and 7.6

Note:

- Each integrity constraint check is expressed as a SPARQL ASK query or query template. When applied to an RDF graph, an ASK query returns true if that graph contains one or more instances of the data cube that violate the constraint (source: https://www.w3.org/TR/vocab-data-cube/#wf-rules)

Tip: The rdflib library can also be used to execute SPARQL

- (see https://rdflib.readthedocs.io/en/stable/intro_to_sparql.html)

References

The RDF Data Cube Vocabulary

- https://www.w3.org/TR/vocab-data-cube/

SDMX

- sdmx-concept: https://raw.githubusercontent.com/UKGovLD/publishing-statistical-data/master/specs/src/main/vocab/sdmx-concept.ttl
- sdmx-dimension: https://raw.githubusercontent.com/UKGovLD/publishing-statistical-data/master/specs/src/main/vocab/sdmx-dimension.ttl
- sdmx-measure: https://raw.githubusercontent.com/UKGovLD/publishing-statistical-data/master/specs/src/main/vocab/sdmx-measure.ttl

Python

- rdflib library: https://rdflib.readthedocs.io/en/stable/
- SPARQL querying: https://rdflib.readthedocs.io/en/stable/intro_to_sparql.html