Overview of SHACL

https://book.validatingrdf.com/bookHtml011.html

SHACL Example

You can test the next two examples (slides) with the following data graph

```
@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 foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex: <http://example.org/> .

ex:Donal_Trump
    a ex:User;
    foaf:name
        "Donald Trump" ;
    ex:birthdate
        "1946-06-14"^^xsd:date;
    foaf:knows
        ex:Joe_Biden.
```

SHACL Example

Every user has exactly one birthdate and the value (birthdate) must be xsd:date.

```
@prefix ex: <http://example.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:UserShape a sh:NodeShape;
        sh:targetClass ex:User;
        sh:property [
                sh:path ex:birthdate;
                sh:minCount 1;
                sh:maxCount 1;
                sh:datatype xsd:date;
        ].
```

- **Prefix**
- **User defined**
- **All nodes that are instances of ex:User**
- **Predicat**
- **Exactly one**
- **Value String**

SHACL Example

The value in a foaf:knows property has to be a URI

```
@prefix ex: <http://example.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:UserShape a sh:NodeShape;
        sh:targetClass ex:User;
        sh:property [
                sh:path foaf:knows;
                sh:nodeKind sh:IRI;
        ].
```

- **Prefix**
- **User defined**
- **All nodes that are instances of ex:User**
- **Predicat**
- **Exactly one**
- **Value String**

Some Concepts

General Concepts

- sh:NodeShape
- sh:targetClass
- sh:property
- sh:path

Others

- sh:class
- sh:minCount
- sh:maxCount
- sh:datatype
- sh:or
- sh:and
- sh:hasValue
- sh:value
- sh:pattern
- sh:in
- sh:nodeKind

Check **table 5.3** under **5.6.3 Constraint Components** in the documentation for more core concepts of SHACL

pySHACL Example

https://pypi.org/project/pyshacl/

```
from pyshacl import validate

results = validate(data_graph,
                   shacl_graph=sg,
                   ont_graph=og,
                   inference='rdfs',
                   abort_on_first=False,
                   allow_infos=False,
                   allow_warnings=False,
                   meta_shacl=False,
                   advanced=False,
                   js=False,
                   debug=False)

conforms, results_graph, results_text = results
```

data_graph - is an rdflib Graph object or file path of the graph to be validated

shacl_graph - is an rdflib Graph object or file path or Web URL of the graph containing the SHACL shapes to validate with

inference - is a Python string value to indicate whether or not to perform OWL inferencing expansion of the data_graph before validation. Options are ***'rdfs', 'owlrl', 'both', or 'none'***. The default is '**none**'.

results - a three-component tuple containing:

- conforms - a bool value
- results_graph - graph
- results_text - string of Validation report

RDFS

INFO216 – Lab 7

RDFS subClassOf

```
g.add((ex.Cade, RDF.type, ex.Student))
```

CADE

← **A (rdf type) Student**

```
g.add((ex.Student, RDFS.subClassOf, FOAF.Person))
```

Every Student is also a Person

⇒ (implies)

CADE

← **A (rdf type) Person**

RDFS subPropertyOf

```
g.add((ex.Cade, ex.studentAt, ex.UIB))
```

CADE

← **Student At UIB**

```
g.add((ex.studentAt, RDFS.subPropertyOf, ex.attends))
```

Someone who is a student somewhere also attends that place

⇒ (implies)

CADE

← **Attends UIB**

RDFS range & domain

```
g.add((ex.Emma, ex.flyTo, ex.Bergen))
```

In a “flyTo” triple, the subject is always a person, and the object is always a city

```
g.add((ex.flyTo, RDFS.domain, FOAF.Person))
g.add((ex.flyTo, RDFS.range, ex.City))
```

```
g.add((ex.Emma, RDF.type, FOAF.Person))
g.add((ex.Bergen, RDF.type, ex.City))
```

THESE ARE ADDED!

⇒ (implies)

A (rdf type) Person →  EMMA

 BERGEN ← **A (rdf type) City**

Like this

TIP: In a domain triple, imagine the subject is replaced with the “original” subject, and the property replaced with rdf:type; and the subject is replaced with the “original” object in a range triple.

RDFS Closure

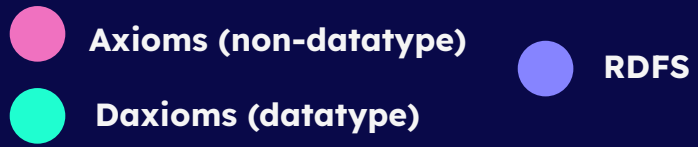
RDFSClosure / DeductiveClosure

The entailments (and axioms) of your graph can be shown with closure.

```
engine = owlrl.RDFSClosure.RDFS_Semantics(graph, False, False, False)
engine.closure()
engine.flush_stored_triples()

OR

owlrl.DeductiveClosure(owlrl.RDFS_Semantics).expand(g)
```



Closure Showcase in VSCode

https://drive.google.com/file/d/1ZirgVmG35LnQAIvWsKwZF6DYMb4nKYUB/

INFO216_Coding > 2023 > Labs > lab7 > lab7_example.py > ...

```python
from rdflib import Graph, RDFS, Namespace, RDF, FOAF, Literal, XSD
import owlrl

g = Graph()
ex = Namespace('http://example.org/')

g.bind("ex", ex)
g.bind("foaf", FOAF)

#populate the graph
g.add((ex.Cade, RDF.type, ex.Student))
g.add((ex.Cade, ex.studentAt, ex.UIB))

g.add((ex.Emma, ex.flyTo, ex.Bergen))

#rules
g.add((ex.Student, RDFS.subClassOf, FOAF.Person))
g.add((ex.studentAt, RDFS.subPropertyOf, ex.attends))

g.add((ex.flyTo, RDFS.domain, FOAF.Person))
g.add((ex.flyTo, RDFS.range, ex.City))


print(g.serialize())

```

OWL 1

Lab 8 – INFO216

Python Example On GitHub

(A)Symmetric Properties

https://www.w3.org/TR/owl2-primer/#Advanced_Use_of_Properties

Symmetric

```
g.add((ex.Emma, ex.neighborTo, ex.Cade))
```


```
g.add((ex.neighborTo, RDF.type,
OWL.SymmetricProperty))
```

“You can swap the subject and object with each other”

Asymmetric

```
g.add((ex.Emma, ex.hasFather, ex.Tom))
```


```
g.add((ex.hasFather, RDF.type,
OWL.AsymmetricProperty))
```

“You cannot swap the subject and object with each other”

(Ir)Reflexive Properties

https://www.w3.org/TR/owl2-primer/#Advanced_Use_of_Properties

Reflexive

g.add((ex.livesWith, RDF.type, OWL.ReflexiveProperty))

“You are related to yourself”

Irreflexive

g.add((ex.Emma, ex.hasFather, ex.Tom))

g.add((ex.hasFather, RDF.type, OWL.IrreflexiveProperty))

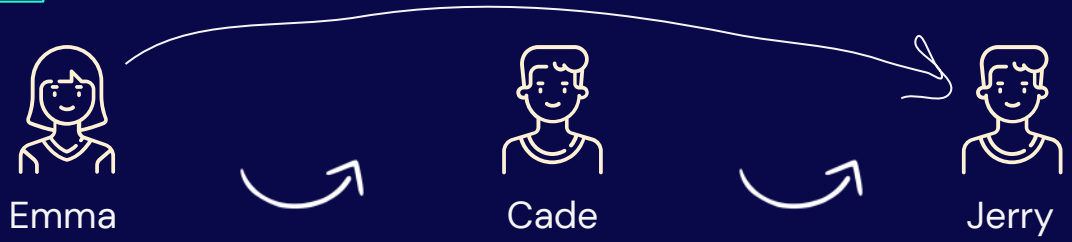
“You cannot be related to yourself”

Transitive Properties

https://www.w3.org/TR/owl2-primer/#Advanced_Use_of_Properties

*"Everything you can do in **two steps**, you can do **in one**"*

```
g.add((ex.Emma, ex.groupPartner, ex.Cade))
g.add((ex.Cade, ex.groupPartner, ex.Jerry))
```




```
g.add((ex.groupPartner, RDF.type, OWL.TransitiveProperty))
```

This gets added!

```
ex:Emma ex:groupPartner ex:Jerry
```

Functional Properties

https://www.w3.org/TR/owl2-primer/#Advanced_Use_of_Properties

Functional

```
g.add((ex.Emma, ex.birthdate,
Literal("1996-10-22", datatype=XSD.date)))

g.add((ex.birthdate, RDF.type,
OWL.FunctionalProperty))
```

"You can only have one instance of it, however it is not unique. Like a birthdate, you only have one, but several people share your birthdate."

InverseFunctional

```
g.add((ex.Emma, ex.socialSecurityNumber,
Literal("123456789", datatype=XSD.integer)))

g.add((ex.socialSecurityNumber, RDF.type,
OWL.InverseFunctionalProperty))
```

"You are the only one with this particular object, no other subject shares this object with you. Like a social security number that only you have, and no one shares it."

Inverse Of

```
g.add((ex.Emma, ex.hasFather, ex.Tom))
```

Emma Tom

TIP:
You can use the ^ to specify the inverse of in SPARQL

```
g.add((ex.hasFather, OWL.inverseOf, ex.fatherOf))
```

Tom Emma

This gets added!

```
ex:Tom ex:fatherOf ex:Emma
```

Differences & Equivalences

Individual	Predicate	Class
OWL.sameAs OWL.differentFrom **GROUPWISE** OWL.AllDifferent OWL.distinctMembers	OWL.equivalentProperty OWL.propertyDisjointWith **GROUPWISE** OWL.AllDisjointProperties OWL.members	OWL.equivalentClass OWL.disjointWith **GROUPWISE** OWL.AllDisjointClasses OWL.members
g.add((ex.Emma, OWL.differentFrom, ex.Cade))	g.add((FOAF.knows, OWL.equivalentProperty, schema.knows))	g.add((ex.Student, OWL.equivalentClass, dbpedia.Student))

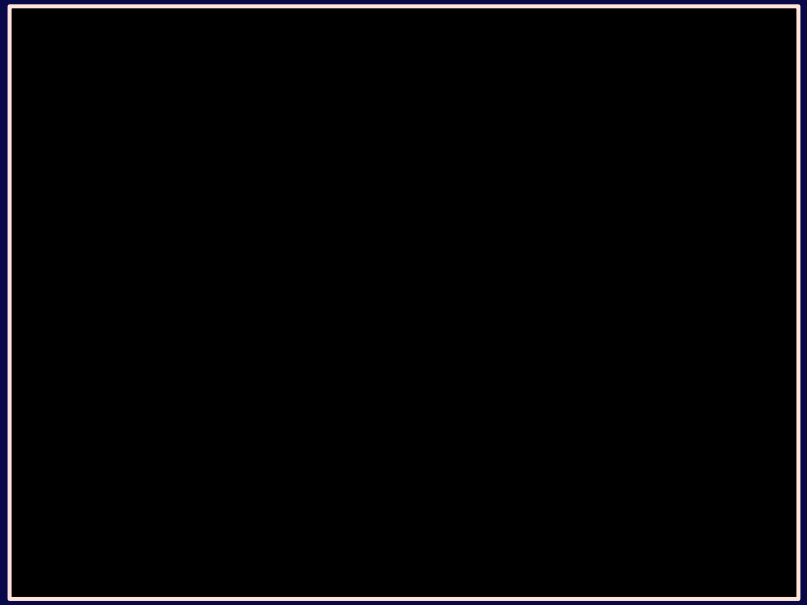
OWL 2 Protégé

Lab 9 - INFO216

Protégé Example on GitHub

Protégé Installation

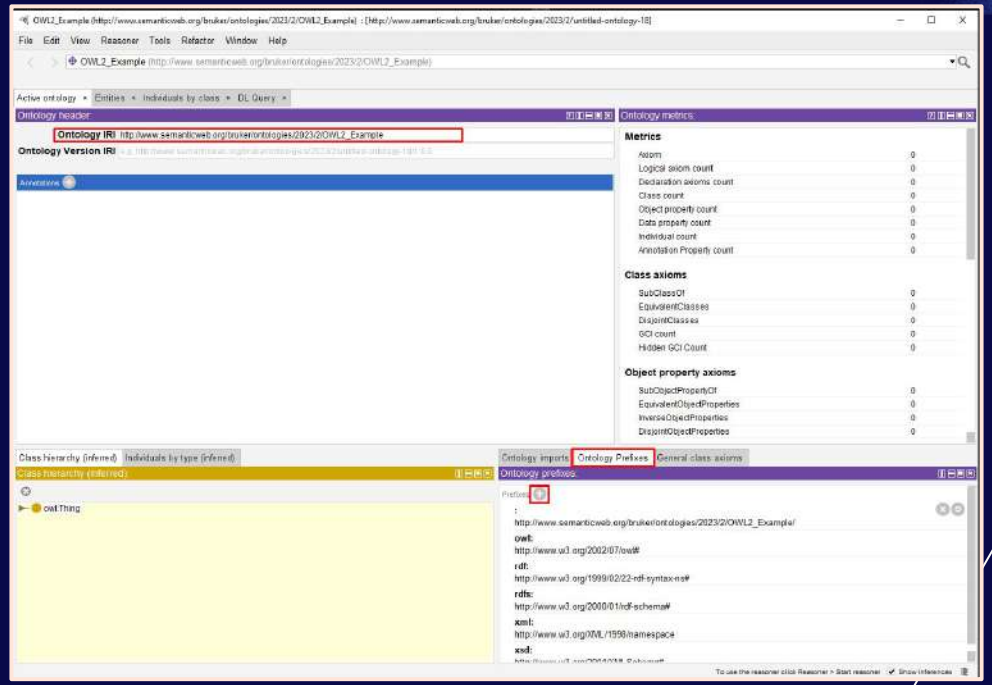
Go to Protégé's website, and follow the instructions for your operating system



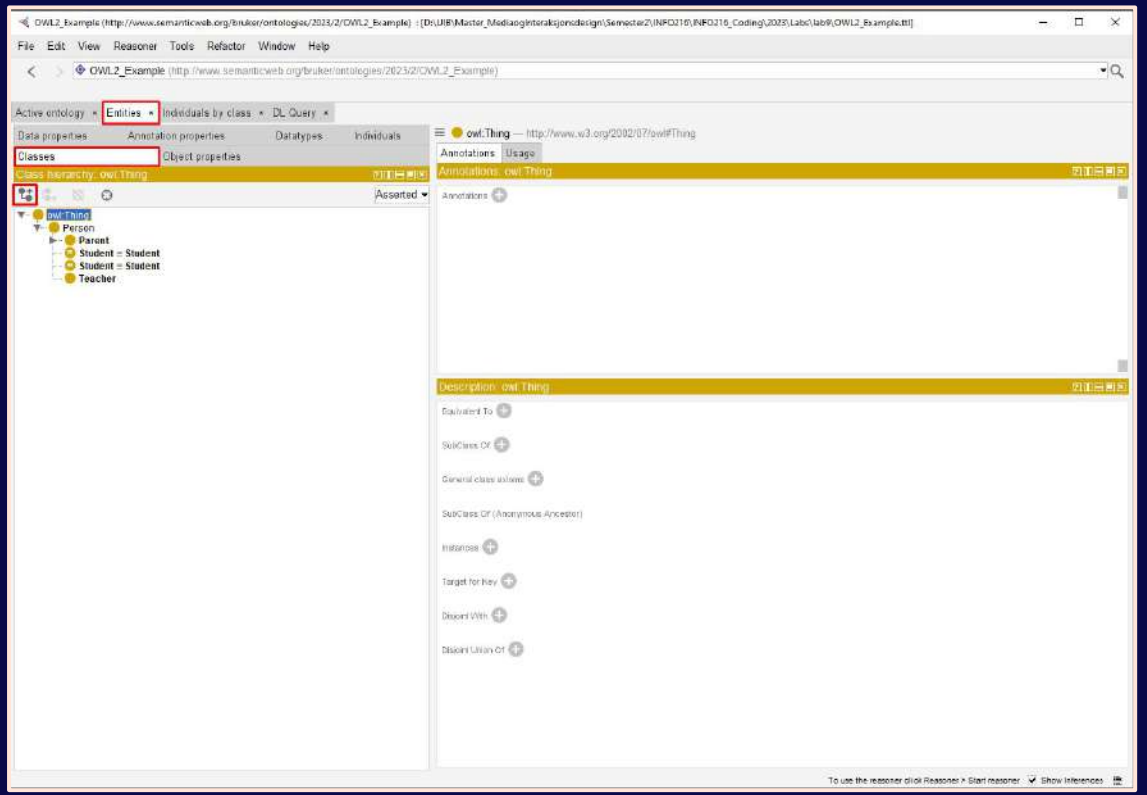
Protégé Name & Prefixes

Under “Active ontology”, you can find the “Ontology IRI” where you can change the name of the ontology by replacing “untitled-ontology-x”; where x is a number.

Under “Ontology Prefixes” you can add new prefixes (e.g. dbpedia / FOAF) by clicking the plus  icon



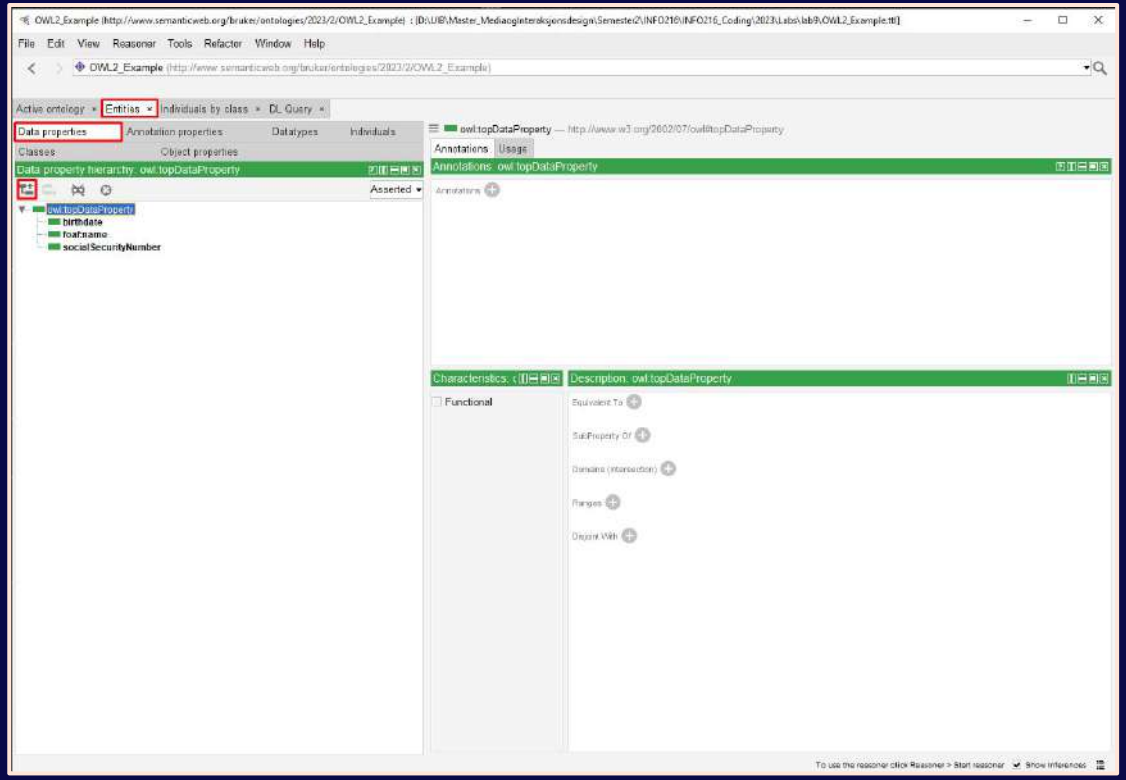
Protégé Classes



Under “Entities” then “Classes” you can create classes and subclasses under “owl:Thing”. To *create* a class click this icon.

To *delete* a class *highlight* the class & click the following icon

When *selecting* a Class, you can see options to specify *characteristics*; e.g. is the class disjoint with another class?

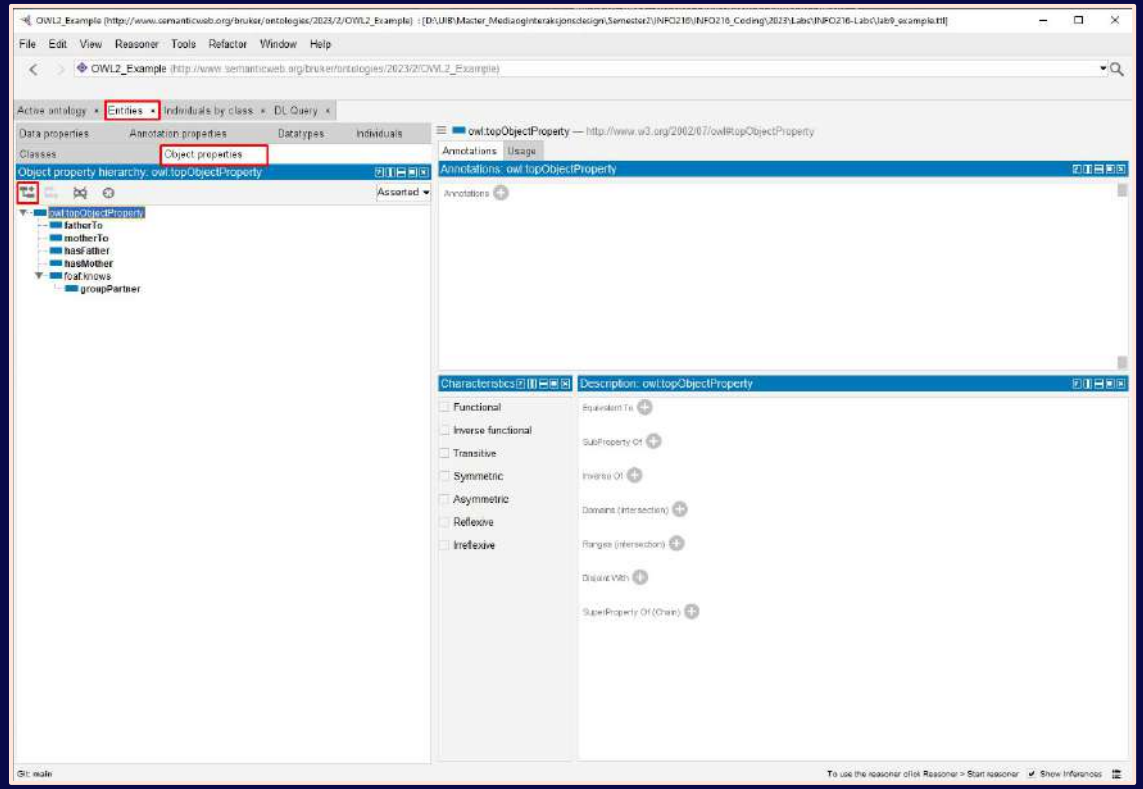


Under “Entities” then “Data properties” you can create properties under “owl:topDataProperty”. To *create* a property click this icon.

To *delete* a property *highlight* the property & click the following icon

When *selecting* a Property, you can see options to specify *characteristics*; e.g. what are the domain (subject) and range (object) of triples with that property?

Protégé Data Properties



Under “Entities” then “Object properties” you can create properties under “owl:topObjectProperty”. To *create* a property click this icon.

To *delete* a property *highlight* the property & click the following icon

When *selecting* a Property, you can see options to specify *characteristics*; e.g. what are the domain (subject) and range (object) of triples with that property?

Protégé Object Properties

Data VS Object Properties

Data Properties

Describes the relation between instances and datavalues.

For instance:
foaf:name rdfs:domain foaf:Person
foaf:name rdfs:range xsd:string

ex:Emma foaf:name “Emma”

Object Properties

Describes the relation between two instances/individuals.

For instance:
ex:teaches rdfs:domain ex:Lecturer
ex:teaches rdfs:range ex:Student

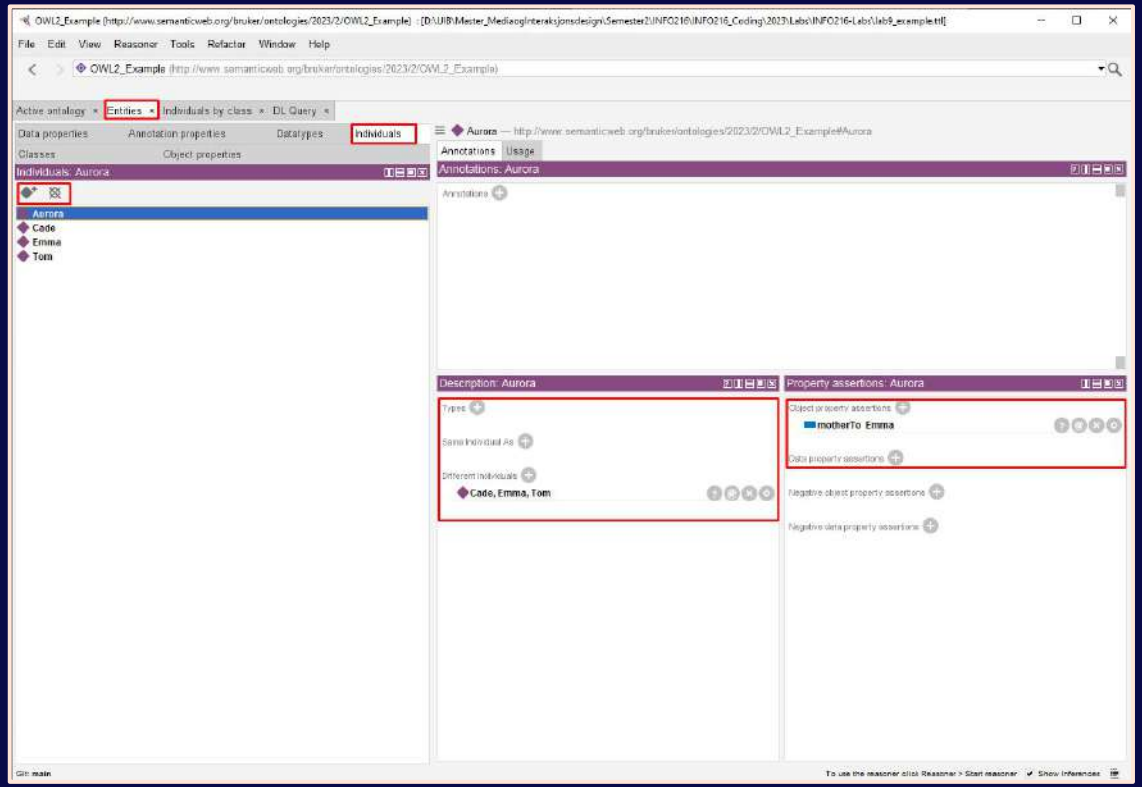
ex:Jeff ex:teaches ex:Emma

Protégé Individuals

Under “Entities” then “Individuals” you can create individuals. To *create* an individual click this  icon.

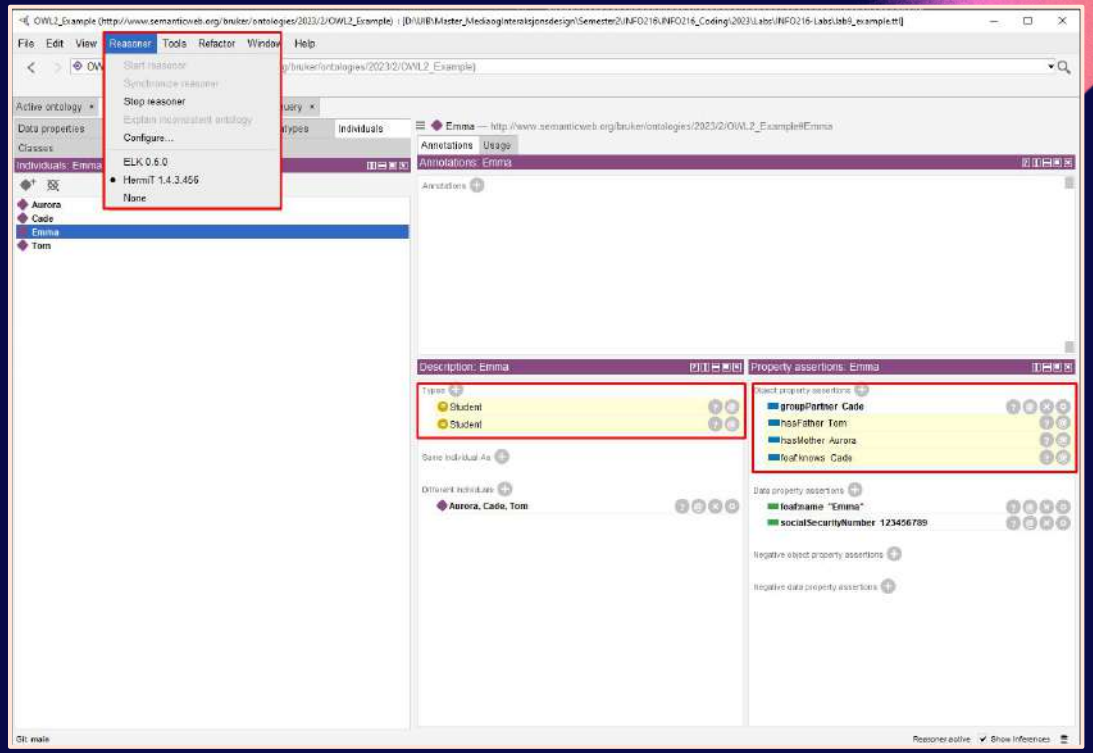
To *delete* an individual *highlight* it & click the following  icon

When *selecting* an individual you can specify characteristics for that individual; for example using the specified object and data properties in your ontology.



Protégé Reasoner

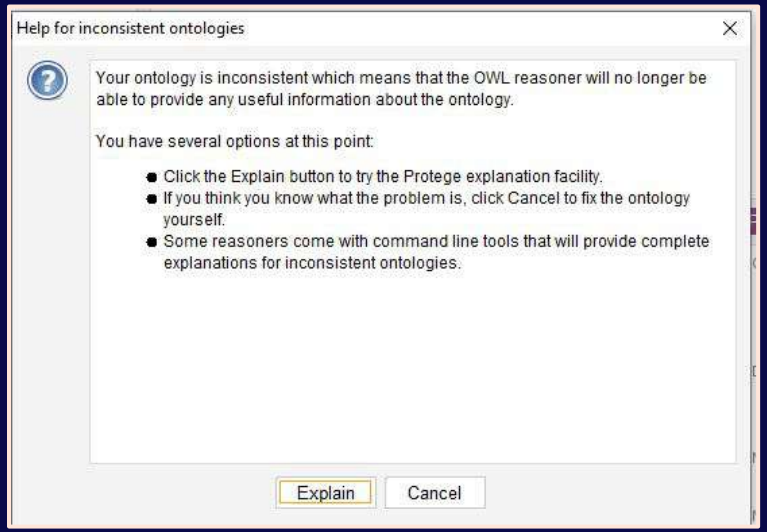
When running the “Reasoner” it will run indefinitely until you turn it off. You can see the axioms and entailments when for example highlighting an individual you’ve created. These axioms and entailments will have a yellow background; they will disappear when you turn of the reasoner.



Protégé Reasoner Inconsistencies

If you have any *inconsistencies* in your ontology the following warning (see image) will appear when running the reasoner.

The inconsistency will also be highlighted in red within Protégé



Tip to understand Protégé

- Open the turtle file I provided in your IDE (e.g. VSCode)
- Pick a random Class, Data Property, Object Property and Individual. E.g. you could look at:
  - Father (Class)
  - foaf:name (Data Property)
  - fatherTo (Object Property)
  - Emma (Individual)
- Try to get an understanding of how they are defined. For example are they a subClassOf/subPropertyOf anything?
- Then browse Protégé and observe how the Class, Properties and Individual are defined in Protégé. For example: how does a Class become a subClassOf something?