

RDFS: RDF Vocabulary Description Language

We would like to define RDF types.

Quelle: <http://www.w3.org/TR/rdf-concepts/>



Expl. cont.

```

ex:MotorVehicle rdf:type rdfs:Class .
ex:PassengerVehicle rdf:type rdfs:Class .
ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
ex:MiniVan rdf:type rdfs:Class .

ex:PassengerVehicle rdfs:subClassOf ex:MotorVehicle .
ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:Truck rdfs:subClassOf ex:MotorVehicle .
ex:MiniVan rdfs:subClassOf ex:Van .
ex:MiniVan rdfs:subClassOf ex:PassengerVehicle .
  
```

RDF vocabulary again

- Classes
rdf:Property, rdf:Statement, rdf:XMLLiteral, rdf:Seq, rdf:Bag, rdf:Alt, rdf>List
- Properties
rdf:type, rdf:subject, rdf:predicate, rdf:object, rdf:first, rdf:rest, rdf:_n, rdf:value
- Resources
rdf:nil

RDFS vocabulary

- Classes
rdfs:Resource, rdfs:Class, rdfs:Literal, rdfs:Datatype, rdfs:Container,
rdfs:ContainerMembershipProperty.
- Properties
rdfs:domain, rdfs:range, rdfs:subPropertyOf, rdfs:subClassOf, rdfs:member, rdfs:seeAlso,
rdfs:isDefinedBy, rdfs:comment, rdfs:label



Examples:

- Classes:
Student rdf:type rdfs:Class
- Class hierarchy:
Student rdfs:subClassOf Person
- Properties:
hasName rdf:type rdf:Property
- Property hierarchies:
hasMother rdfs:subPropertyOf hasParent
- Associating properties with classes:
hasname rdfs:domain Person
- Associating properties with classes:
hasName rdfs:range xsd:string



Examples cont.:

- Comment:
Person rdfs:comment "A person is any human being"
- Label:
Person rdfs:label "Human being"
- see also:
person rdfs:seeAlso <http://xmlns.com/wordnet/1.6/Human>
- is defined by:
person rdfs:isDefinedBy <http://xmlns.com/wordnet/1.6/Human>

Entailment rules

- RDFS semantics is defined by entailment rules.
- If an RDFS graph S , i.e. an RDF graph extended by statements referring to the RDFS vocabulary, contains triples on which such rules apply, then additional triples are added to S .
- After having applied entailment rules in all possible ways, we have produced the *realization* of S .
- S entails an RDF graph E , if E is a subgraph of the realization of S .

Some interesting entailment rules

- $(a, \text{rdf:type}, \text{rdf:property}) \longrightarrow (a, \text{rdfs:subproperty}, a)$
- $(a, \text{rdfs:subproperty}, b), (b, \text{rdfs:subproperty}, c) \longrightarrow (a, \text{rdfs:subproperty}, c)$
- $(a, \text{rdfs:subproperty}, b), (c, a, d) \longrightarrow (c, b, d)$
- $(a, \text{rdf:type}, \text{rdf:class}) \longrightarrow (a, \text{rdfs:subclass}, a)$
- $(a, \text{rdfs:subclass}, b), (b, \text{rdfs:subclass}, c) \longrightarrow (a, \text{rdfs:subclass}, c)$
- $(a, \text{rdfs:subclass}, b), (c, \text{rdf:type}, a) \longrightarrow (c, \text{rdf:type}, b)$
- $(a, \text{rdfs:domain}, b), (c, a, d) \longrightarrow (c, \text{rdf:type}, b)$
- $(a, \text{rdfs:range}, b), (c, a, d) \longrightarrow (d, \text{rdf:type}, b)$

Abbildung relationaler Datenbanken nach RDF

direkte Abbildung

- Relationsschema \longrightarrow Klasse
- Tupel einer Relation \longrightarrow Objekt (Resource)
- Attribut \longrightarrow Eigenschaft (Property)
- Attribut-Mapping \longrightarrow Kante
- Wert \longrightarrow Literal oder Objekt (Resource)

Key, foreign key and other constraints?

These constraints should not be lost because otherwise problems arise

- if a user builds her own knowledge base by integrating several RDF graphs found on the internet,
- if an exported RDF graph is imported in a relational database at another place,
- if updates on a materialized RDF graph have to be performed such that key and foreign key properties and other constraints have to be checked.

Relational Databases

- A relational database schema \mathbf{R} is a set of relation schemata identified by R , $\mathbf{R} = (R_1, \dots, R_n)$.
 $Att(R)$ denotes the set of attributes of the relation symbol R .
An instance I of \mathbf{R} is a tuple (I_1, \dots, I_n) , where for $i \in [n]$ I_i is a finite instance of R_i , i.e. a finite subset of the n -ary cartesian product over an underlying domain.
An element $\mu \in I$ is called tuple.

A simple mapping RDB to an RDF-graph

Let R be a relation schema, where $Att(R) = A_1, \dots, A_k$.

- Introduce a class C_R and properties $Q_{R,A_1}, \dots, Q_{R,A_k}$.
- Let I be an instance of R .
- For every tuple $\mu = (a_1, \dots, a_k) \in I$ introduce a unique blank node $_n\mu$ and a labelled edge $(_n\mu, rdf : type, C_R)$.
- For every nonnull value $\mu.A$ of μ , $A \in Att(R)$, introduce an edge $(_n\mu, Q_{R,A}, (\mu.A)_{_n\mu, Q_{R,A}})$.
If $\mu.A$ is a key-value, then $(\mu.A)_{_n\mu, Q_{R,A}}$ is a URI; otherwise $\mu.A$ is a literal.

Constraint Checking by SPARQL

- Checking by ASKing for a violation of a respective constraint.
- This differs from SQL, where we ask for fulfillment of a respective constraint.
- This is because SPARQL does not provide a NOT-operator in front of a general expression, what is possible in SQL.

Key constraints:
do p_1, \dots, p_n form a key for the instances of class C ?

```
ASK {  
  ?x rdf:type C.  
  ?y rdf:type C.  
  ?x p1 ?p1; [...]; pn ?pn.  
  ?y p1 ?p1; [...]; pn ?pn.  
  FILTER (?x!=?y)  
}
```

Foreign key constraints:
do p_1, \dots, p_n form a foreign key with respect to key q_1, \dots, q_n ?

```
ASK {  
  ?x rdf:type C; p1 ?p1; [...]; pn ?pn.  
  OPTIONAL {  
    ?y rdf:type D; q1 ?p1; [...]; qn ?qn.  
  } FILTER (!bound(?y))  
}
```

max-Cardinality:
Does p have at most n different values?

```
ASK {
  ?x rdf:type C.
  ?x p ?p1; [...]; p ?pn+1.
  FILTER (allDist([?p1, ..., ?pn+1]))
}
```

$\text{allDist}([?p1, \dots, ?pn]) \stackrel{\text{def}}{=} \bigwedge_{1 \leq i \leq n} (\bigwedge_{i < j \leq n} ?pi \neq ?pj)$

enforces that variables $?p1, \dots, ?pn$ are pairwise distinct.

min-Cardinality:
Does p have at least n different values?

```
ASK {
  ?x rdf:type C.
  OPTIONAL {
    ?y rdf:type C.
    ?y p ?p1; [...]; p ?pn.
    FILTER (allDist(?p1, ..., ?pn) && ?x=?y)
  } FILTER (!bound(?y))
}
```


Path-constraint:

Is the composition of predicates p_1, \dots, p_n contained in q ?

```
ASK {
  ?x rdf:class C; p1 ?p1.
  ?p1 p2 ?p2. [...]. ?pn-1 pn ?pn.
  OPTIONAL { ?x q ?q. FILTER (?pn=?q) }
  FILTER (!bound(?q))
}
```

**Subclass and Subproperty-constraint**

```
ASK {
  ?x1 rdf:type C.
  OPTIONAL { ?x1 rdf:type D. }
  FILTER (!bound(?x1))
}
```

```
ASK {
  ?x1 p ?x2.
  OPTIONAL { ?x1 q ?x2. }
  FILTER (!bound(?x1))
}
```



Intersection-constraint:

Is class C the intersection of classes D1 and D2?

```

ASK {
  { ?x1 rdf:type :C .
    OPTIONAL { ?y1 rdf:type :D1.
              ?y1 rdf:type :D2.
              FILTER(?x1 = ?y1)
            }
    FILTER(!bound(?y1))
  }
UNION
  { ?x1 rdf:type :D1 .
    ?x1 rdf:type :D2.
    OPTIONAL { ?y1 rdf:type :C.
              FILTER(?x1 = ?y1)
            }
    FILTER(!bound(?y1))
  }
}

```



Union-constraint:

Is class C the union of classes D1 and D2?

```

ASK {
  { ?x1 rdf:type :C .
    OPTIONAL { ?y1 rdf:type :D1. FILTER(?x1 = ?y1) } .
    OPTIONAL { ?y2 rdf:type :D2. FILTER(?x1 = ?y2) } .
    FILTER(!bound(?y1) && !bound(?y2))
  }
UNION
  {
    { ?y1 rdf:type :D1 .
      OPTIONAL { ?x1 rdf:type :C. FILTER(?x1 = ?y1) } }
    UNION
    { ?y2 rdf:type :D2 .
      OPTIONAL { ?x1 rdf:type :C. FILTER(?x1 = ?y2) } }
    FILTER(!bound(?x1))
  }
}

```

