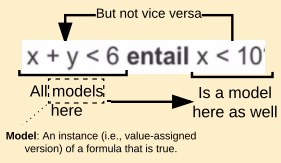
WEEK 1: Entailment and other concepts

ENTAILMENT IN ARITHMETIC FORMULAS



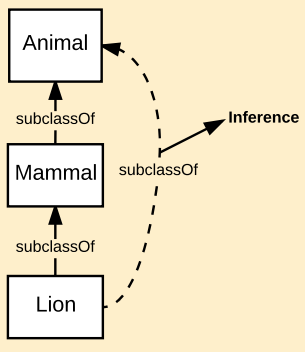
Model: An instance (i.e., value-assigned version) of a formula that is true.

ENTAILMENT IN CONCEPT HIERARCHIES

Concept: An abstraction or generalization arrived through experiences or transformation of existing ideas.

Lion subclassOf Mammals

Everything that is true for mammals must be true for all lions, but not everything true for lions is true for all mammals.



Universe:
{object1, object2, object3, ...}

Knowledge base:
{axiom1, axiom2, axiom3, ...}

Axiom:
Lion subclassOf Mammal

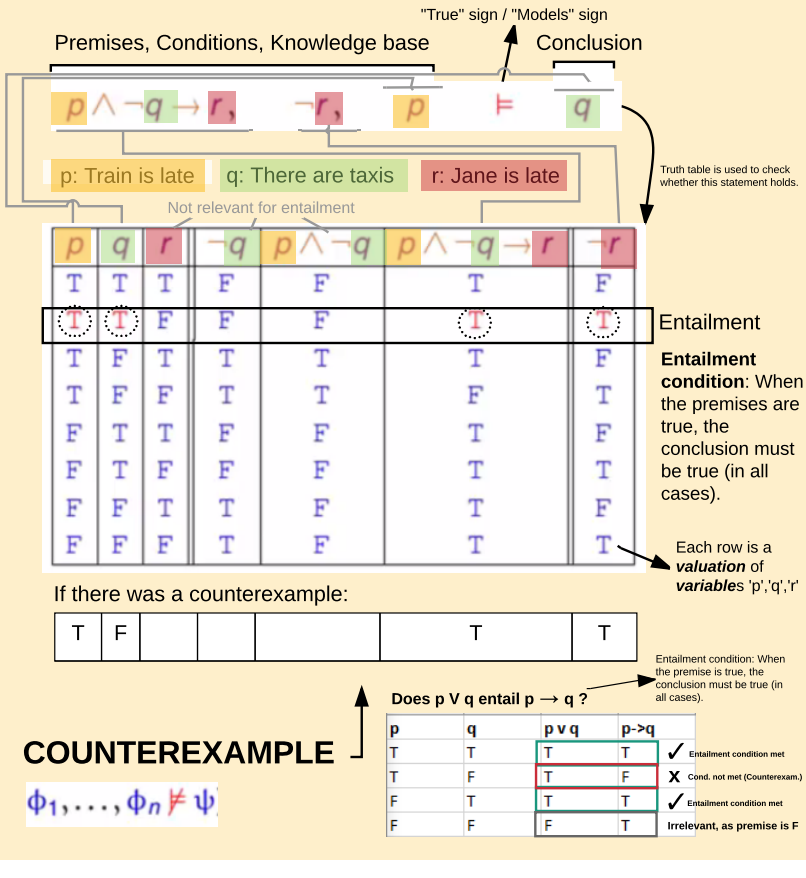
This statement becomes a *model* of the *axiom* only if the model is true,.

Model:
lionNamedX subclassOf Mammals

Not a model (and not a counterexample):
snakeNamedY subclassOf Mammals

This statement is not true; therefore, it is not a model. If it was true, then it would have been a counterexample, but it is not.

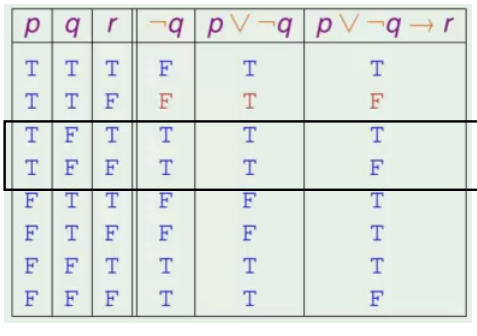
ENTAILMENT IN PROPOSITIONAL LOGIC



Entailment condition: When the premises are true, the conclusion must be true (in all cases).

TRUTH TABLE

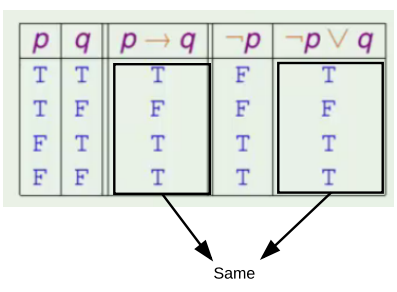
Not all lines in a truth table make sense. Establishing validity is to ascertain which ones do.



e.g., Train is late (p), and there are no taxis (-q), but in one case Jane is late (r) and in the other she is not late.

EQUIVALENCE

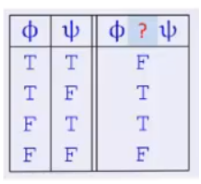
$p \rightarrow q \equiv \neg p \vee q$



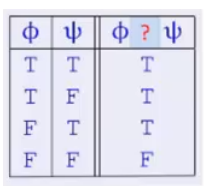
COUNTEREXAMPLE

$\phi_1, \ldots, \phi_n \not\models \psi$

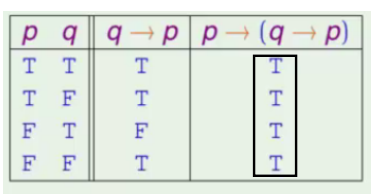
EXCLUSIVE 'OR'



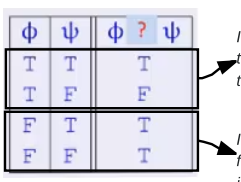
DISJUNCTION ('OR')



TAUTOLOGY



IMPLICATION

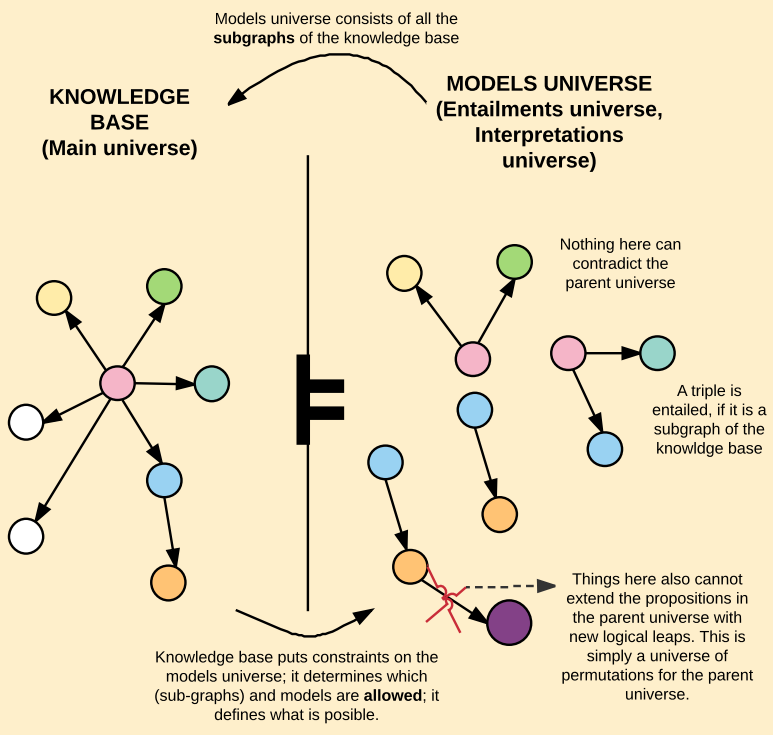


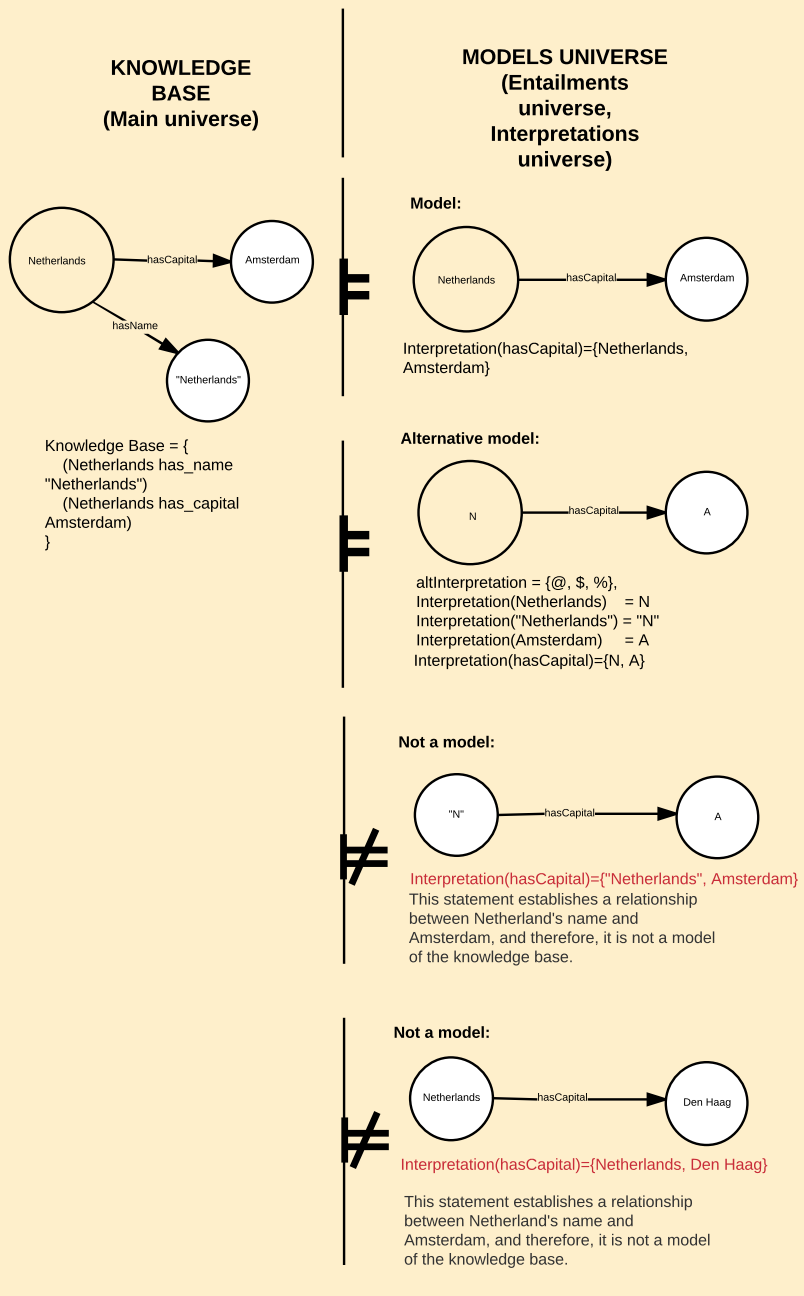
*If you start out with a true premise, then the implication should be true only when the conclusion is also true. (This corresponds to the scenario in when **ϕ** is true, the truth of the implication is the same as the truth of **ψ**.)*

*If you start out with a false premise, then, as far as implication is concerned, you are free to conclude anything. (This corresponds to the scenario in when **ϕ** is false, the implication **ϕ**→**ψ** is true no matter what **ϕ** is.)*

CONTRADICTION

ENTAILMENT IN SIMPLE GROUNDED GRAPHS

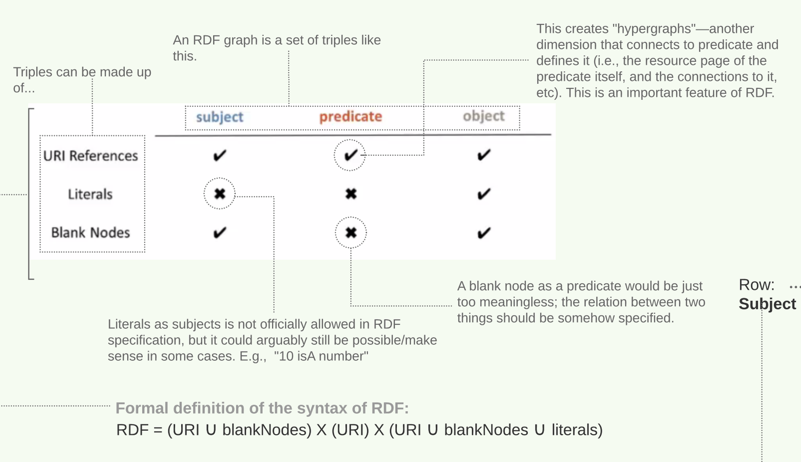




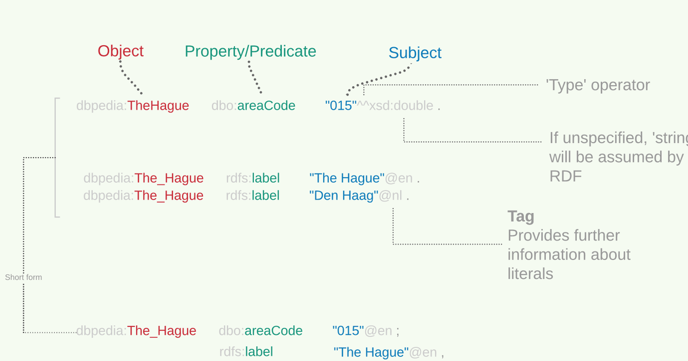
RDF & RDFS
Basics, Semantics, Schema, Vocabulary, and Usage

BASICS

Anatomy

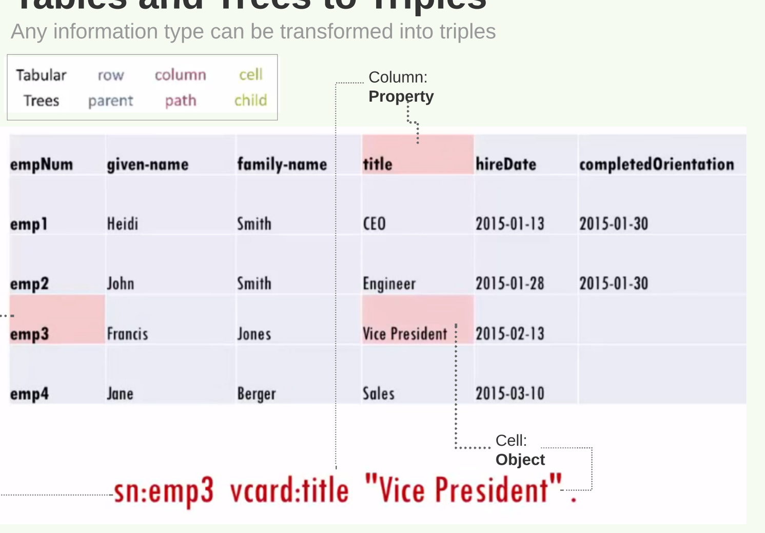


Formal definition of the syntax of RDF:
RDF = (URI ∪ blankNodes) X (URI) X (URI ∪ blankNodes ∪ literals)

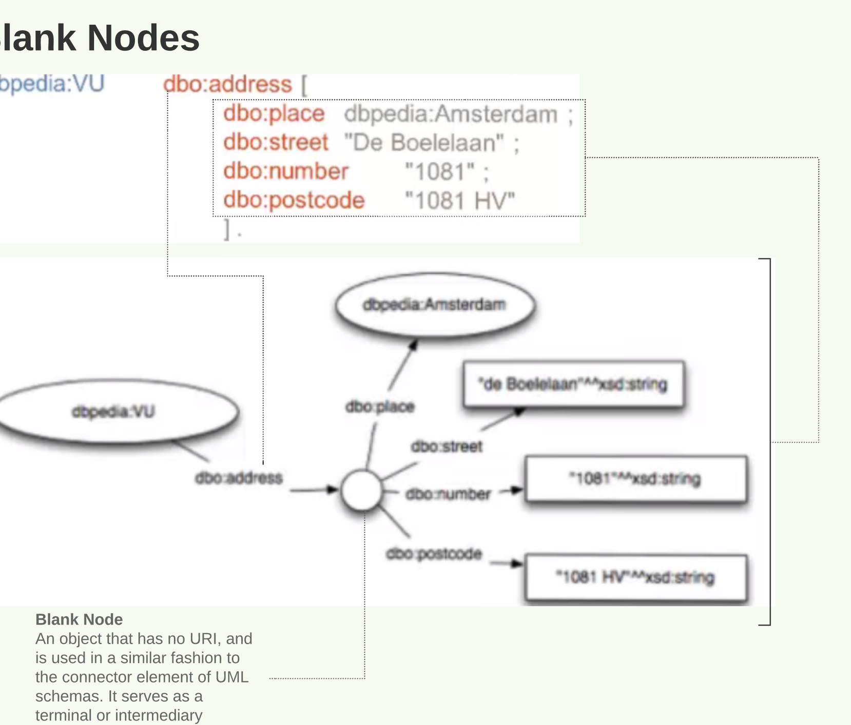


Tables and Trees to Triples

Any information type can be transformed into triples



Blank Nodes



Blank Node
An object that has no URI, and is used in a similar fashion to the connector element of UML schemas. It serves as a terminal or intermediary between two or more elements.

Use them when resource is unknown, or has no natural identifier—when we cannot/don't want to give a name to an object

Example RDF (and Vocabularies)

```
@prefix vcard: <http://www.w3.org/2006/vcard/ns#>.
@prefix sn:    <http://www.snee.com/hr/>.

sn:emp1 vcard:given-name "Heidi".
sn:emp1 vcard:family-name "Smith".
sn:emp1 vcard:title "CEO".
sn:emp1 sn:hireDate "2015-01-13".
sn:emp1 sn:completedOrientation "2015-01-30".

sn:emp2 vcard:given-name "John".
sn:emp2 vcard:family-name "Smith".
sn:emp2 vcard:title "Engineer".
sn:emp2 sn:hireDate "2015-01-28".
sn:emp2 sn:completedOrientation "2015-01-30".

sn:emp3 vcard:given-name "Francis".
sn:emp3 vcard:family-name "Jones".
sn:emp3 vcard:title "Vice President".
sn:emp3 sn:hireDate "2015-02-13".

sn:emp4 vcard:given-name "Jane".
sn:emp4 vcard:family-name "Berger".
sn:emp4 vcard:title "Sales".
sn:emp4 sn:hireDate "2015-03-10".
```

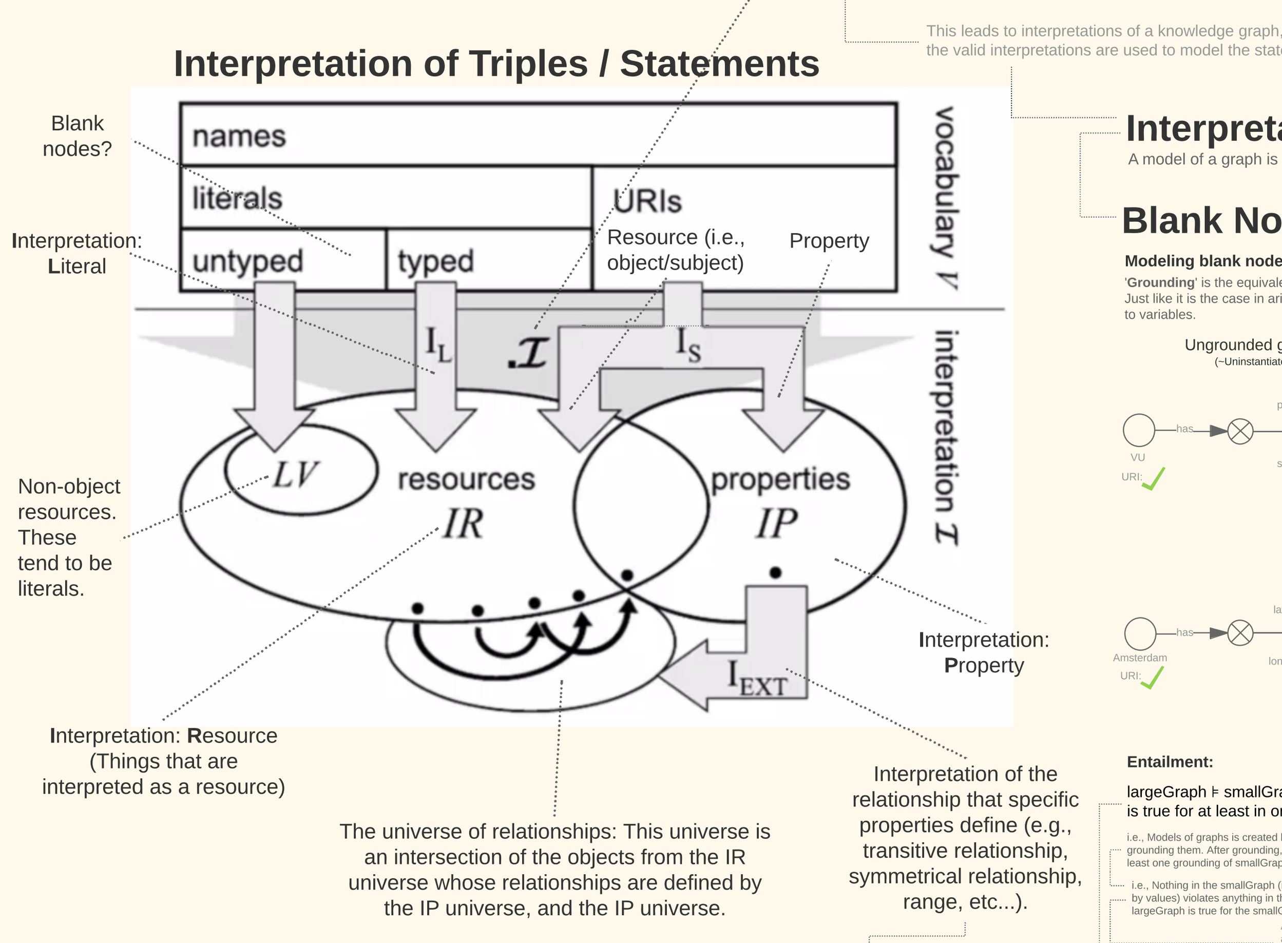
Official w3.org vocabulary
User-created, custom vocabulary

SEMANTICS

Interpretation of each element of a triple/statement.

This leads to interpretations of a knowledge graph, and the valid interpretations are used to model the statement.

Interpretation of Triples / Statements



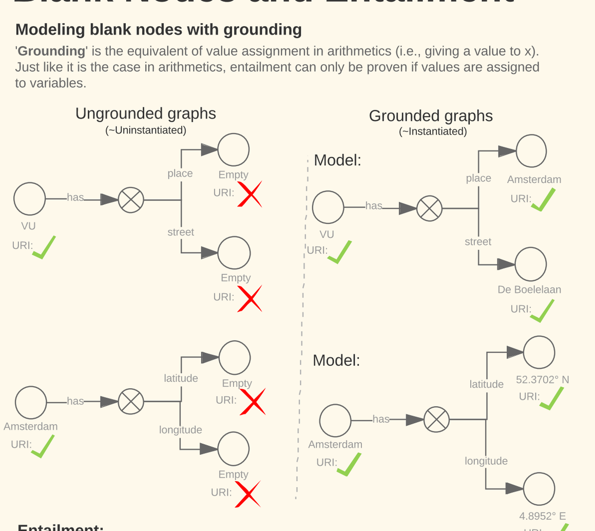
This specificity of properties is not the case in Simple Knowledge Graph Logic (SKGL); but is the case in RDF. SKGL creates graphs while RDF creates Hypergraphs—graphs in which properties are also known as objects and subjects while their properties is being defined.

Interpretation and Models

A model of a graph is an interpretation that satisfies all its triples.

Blank Nodes and Entailment

Modeling blank nodes with grounding
'**Grounding**' is the equivalent of value assignment in arithmetics (i.e., giving a value to x). Just like it is the case in arithmetics, entailment can only be proven if values are assigned to variables.



Entailment:
largeGraph ⊨ smallGraph if and only if largeGraph ⊨ smallGraph(grounded) is true for at least one grounded instance of the smallGraph

i.e., Models of graphs is created by assigning URIs to blank nodes, and thus by grounding them. After grounding, if all models of largeGraph is also a model of at least one grounding of smallGraph, then largeGraph entails smallGraph.

i.e., Nothing in the smallGraph (i.e., a version of it that has its variables replaced by values) violates anything in the largeGraph; and everything that is true in the largeGraph is true for the smallGraph.

~Ungrounded: lion as species
~Grounded: Lion as specimen

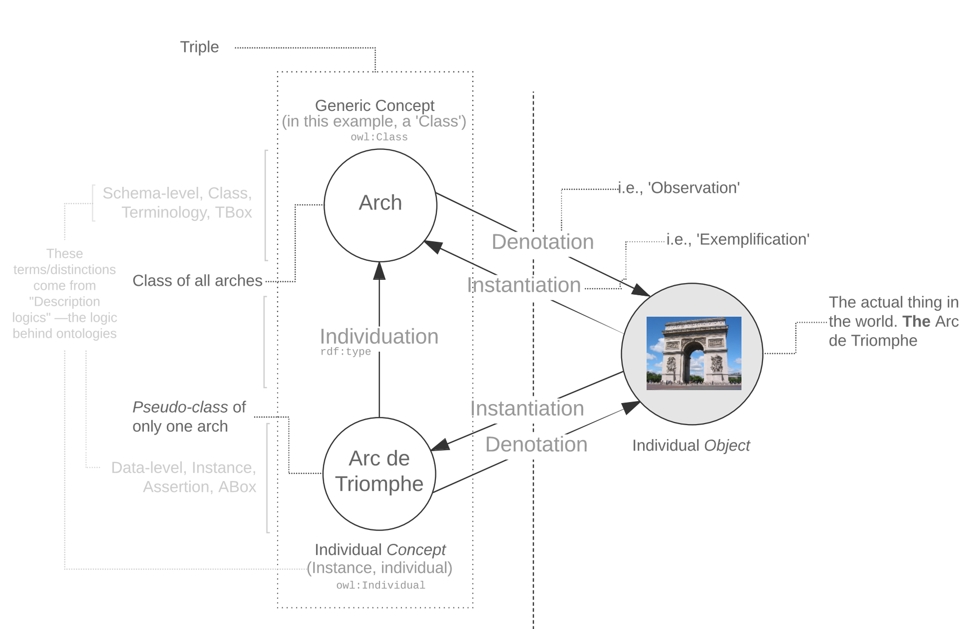
Analogous to saying that nothing that applies to a specimen of a lion violates anything that applies to mammals; and everything that is true for mammals is true for (at least) this sample specimen, and thus, it must be true for mammals. Mammals entail this lion specimen; thus, mammals entail lion species.

From the instructors: Theorem (calculus): A graph G(grounded) is entailed by a graph G if G(grounded) can be rewritten into a subset of G. where rewriting means assigning a URI to a blank node.

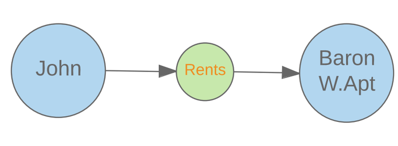
SCHEMA

Knowledge Representation

Class, Type, Instance, Individual, Denotation, Concept, Object

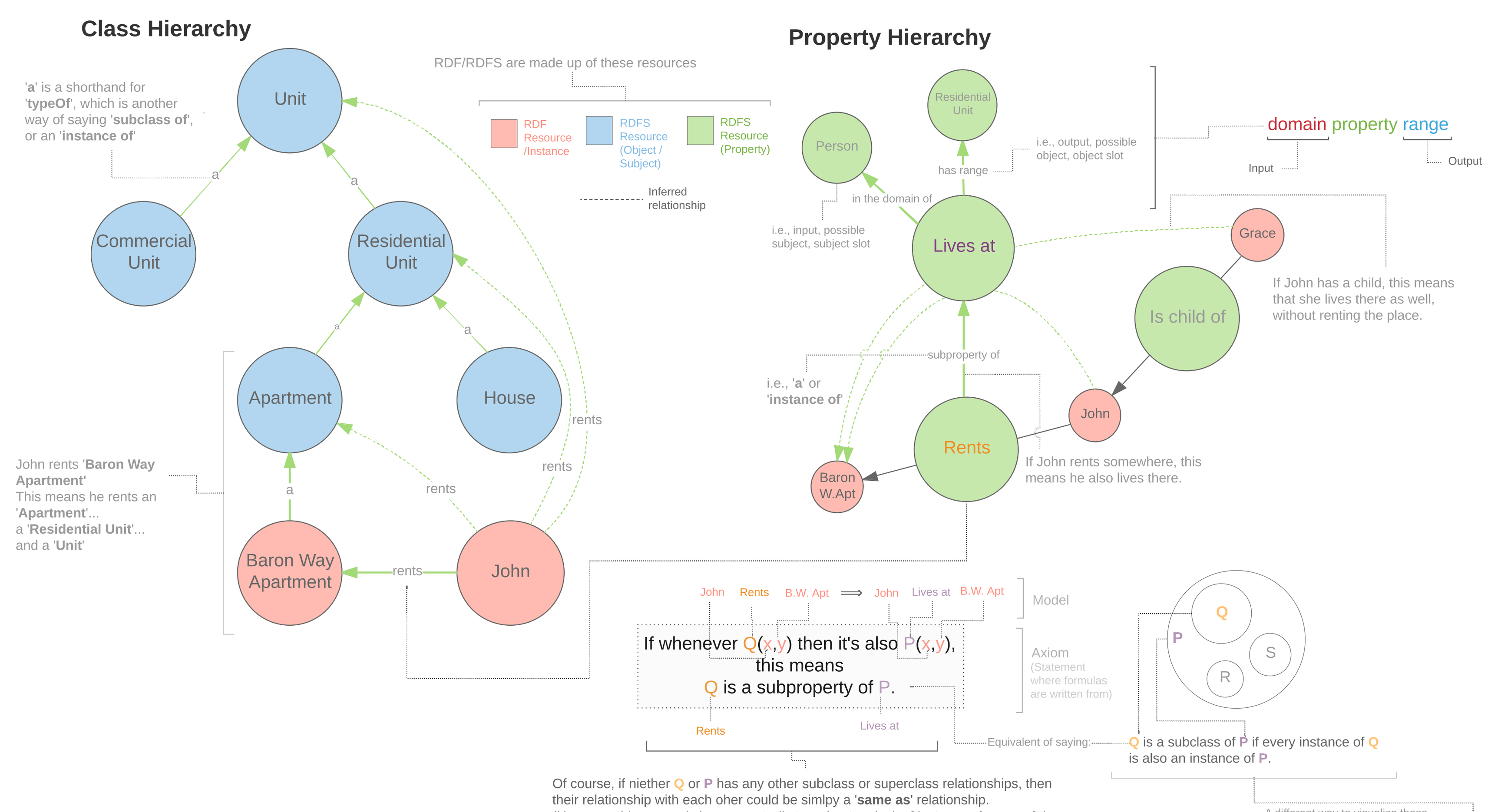


RDF



RDF Schema (RDFS)

RDFS is a basic ontology for formally defining relationships in RDF. It is more limited, but also more efficient (e.g., when calculating entailments) than full-fledged ontology languages such as OWL.



John Rents B.W. Apt ⟹ John Lives at B.W. Apt — Model

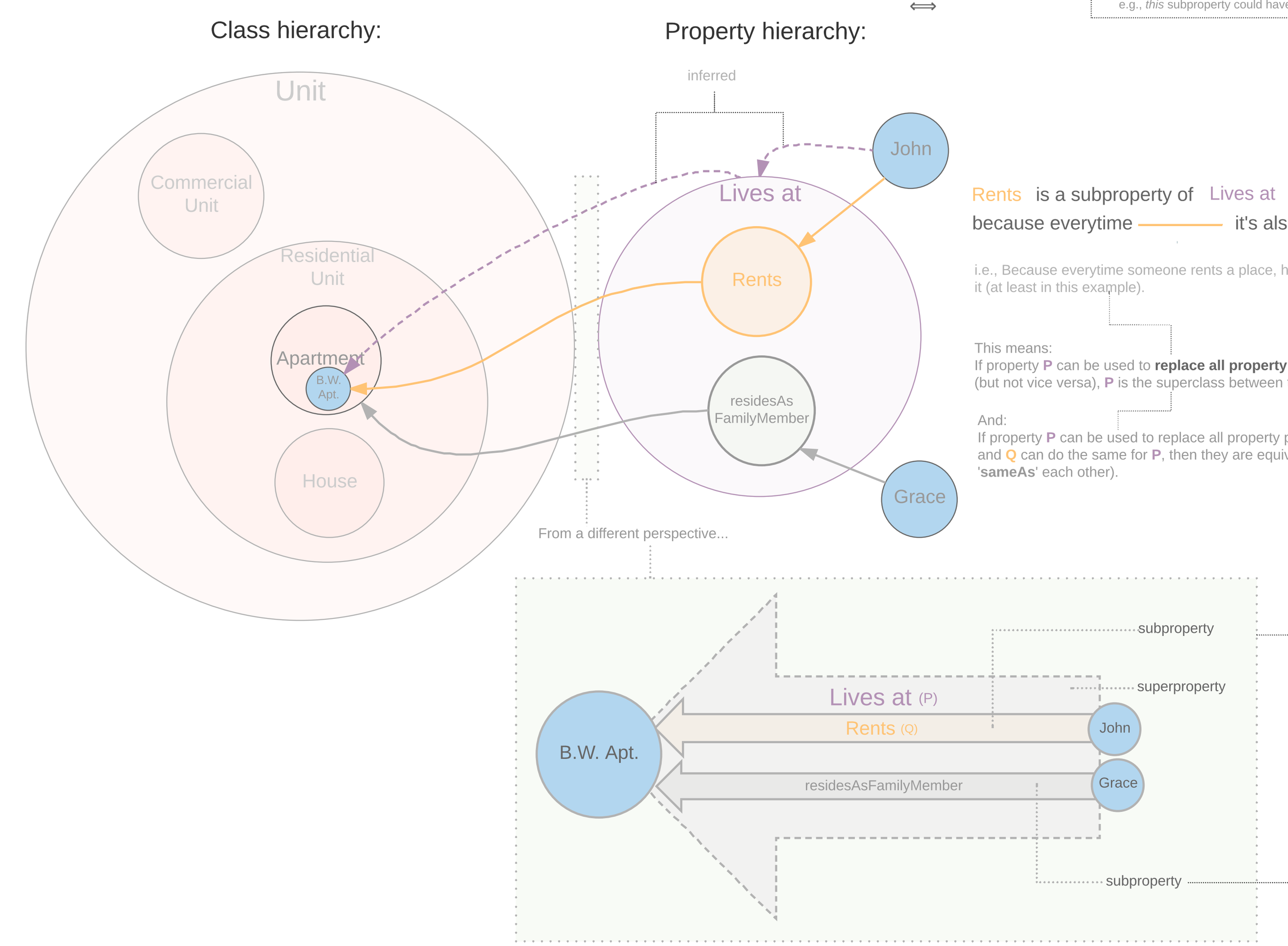
If whenever Q(x,y) then it's also P(x,y), this means Q is a subproperty of P. — Axiom (Statement where formulas are written from)

Equivalent of saying: Q is a subclass of P if every instance of Q is also an instance of P.

A different way to visualize these property hierarchies...

Of course, if neither Q or P has any other subclass or superclass relationships, then their relationship with each other could be simply a 'same as' relationship. (However, this may only be momentarily true due to a lack of instances for one of the subclasses, which would make it empty and perhaps even undefined/undetected.)

e.g., this subproperty could have been momentarily empty



Rents is a subproperty of Lives at because everytime ——— it's also - - - -

i.e., Because everytime someone rents a place, he also lives in it (at least in this example).

This means:
If property P can be used to replace all property paths of Q (but not vice versa), P is the superclass between the two. ——'subclassOf'

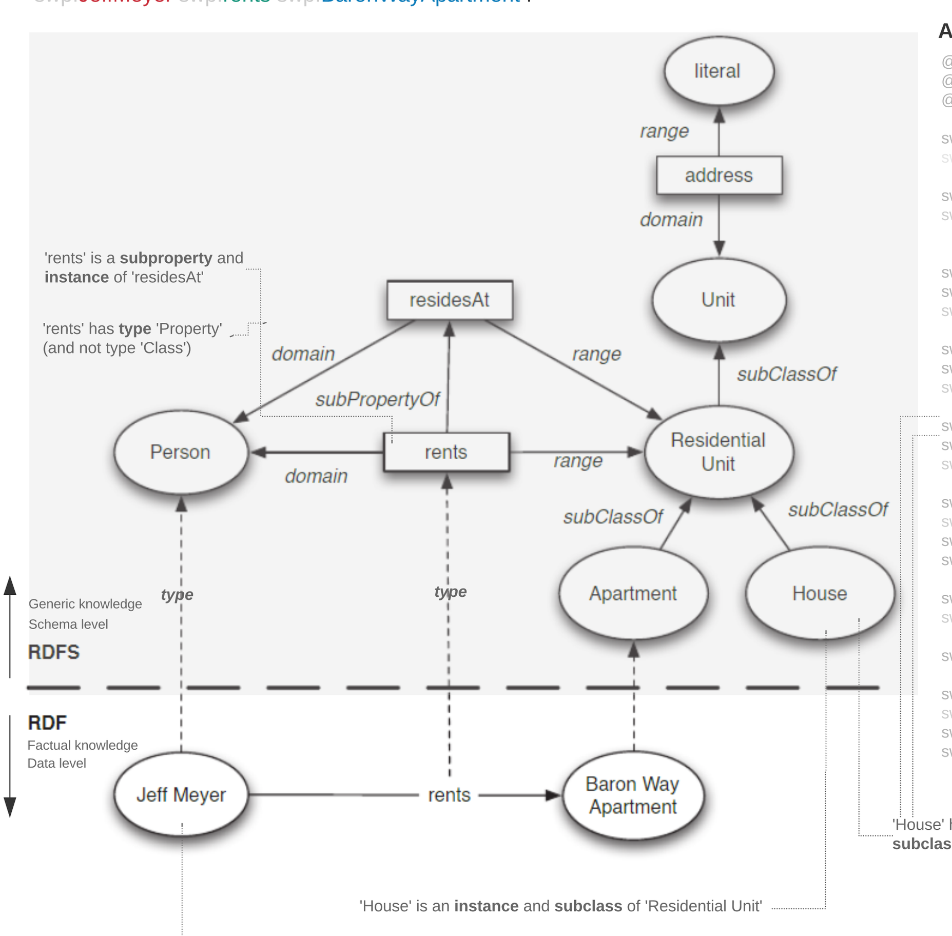
And:
If property P can be used to replace all property paths of Q, and Q can do the same for P, then they are equivalent (i.e., 'sameAs' each other). ——'sameAs'

RDFS Example

An example RDF statement

@prefix swp:<http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.

swp:JeffMeyer swp:rents swp:BaronWayApartment .



'House' is an **instance** and **subclass** of 'Residential Unit'

'Jeff Meyer' or 'Baron Way Aparment' is an **instance** (but **not** a **subclass**) of 'Person'

An example RDFS

@prefix swp:<http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#>.

```
swp:Person  rdf:type  rdfs:Class .
swp:Person  rdfs:comment  "The class of people" .

swp:Unit  rdf:type  rdfs:Class .
swp:Unit  rdfs:comment  "A self contained section of accommodations in a larger building or group of buildings." .

swp:ResidentialUnit  rdf:type  rdfs:Class .
swp:ResidentialUnit  rdfs:subClassOf  swp:Unit .
swp:ResidentialUnit  rdfs:comment  "The class of all units or places where people live." .

swp:Apartment  rdf:type  rdfs:Class .
swp:Apartment  rdfs:subClassOf  swp:ResidentialUnit .
swp:Apartment  rdfs:comment  "The class of apartments" .

swp:House  rdf:type  rdfs:Class .
swp:House  rdfs:subClassOf  swp:ResidentialUnit .
swp:House  rdfs:comment  "The class of houses" .

swp:residesAt  rdf:type  rdfs:Property .
swp:residesAt  rdfs:comment  "Relates persons to their residence" .
swp:residesAt  rdfs:domain  swp:Person .
swp:residesAt  rdfs:range  swp:ResidentialUnit .

swp:rents  rdf:type  rdfs:Property .
swp:rents  rdfs:comment  "It inherits its domain (swp:Person) and range (swp:ResidentialUnit) from its superproperty (swp:residesAt)." .

swp:rents  rdfs:subPropertyOf  swp:residesAt .

swp:address  rdf:type  rdfs:Property .
swp:address  rdfs:comment  "Is a property of units and takes literals as its value" .
swp:address  rdfs:domain  swp:Unit .
swp:address  rdfs:range  rdf:Literal .
```

'House' has **type** 'Class' and is a **subclassOf** 'Residential Unit'

'**type**' property defines the relationship between **class** and **instance**.

'**subclass**' property defines the relationship between **class** and **superclass** (and not between class and instance).

Example in Action: Class-based (weak) Inference with RDFS

Turtle File:

```
@prefix fb: <https://graph.facebook.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

fb:594486635      fb:likes      fb:289482145973 , fb:109327925759703.

fb:109327925759703  fb:name      "Waking Life"@en ;
                    fb:category  fb:movie .

fb:32985985640    fb:name      "Zeitgeist"@en ;
                  fb:category  fb:community .

fb:movie          rdfs:label   "Movie"@en .
fb:community      rdfs:label   "Community"@en .

fb:community      rdfs:subClassOf    fb:movie .
fb:category       rdfs:subPropertyOf rdf:type .
```

This triple allowed Zeitgeist to be inferred as a movie (as it specified that each community is also a movie). Without this line, only 'Waking Life' would have returned.

SPARQL Query:

```
PREFIX fb: <https://graph.facebook.com/>

SELECT ?title WHERE {
    ?movie a      fb:movie ;
           fb:name ?title .
}
```

Query Result:

?title
"Waking Life"@en
"Zeitgeist"@en

VOCABULARY

RDF, RDFS and Other Vocabularies

Resource Description Framework
Resource Description Framework Schema

prefix rdf <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix rdfs <http://www.w3.org/2000/01/rdf-schema#>

RDF/RDFS Classes

Core Classes	Container Classes
rdfs:Resource	rdf:Bag
rdfs:Class	rdf:Seq
rdfs:Literal	rdf:Alt
rdfs:Property	rdfs:Container
rdfs:Statement	
rdfs:Datatype	

RDF/RDFS Properties

Defining Relationships	Restricting Properties	Utility	Reification
rdf:type	rdfs:domain	rdfs:seeAlso	rdf:subject
rdfs:subClassOf	rdfs:range	rdfs:isDefinedBy	rdf:predicate
rdfs:subPropertyOf		rdfs:comment	rdf:object
		rdfs:label	

geo:Amsterdam rdf:type rdf:Resource .
geo:containedIn a rdf:Property .
rdf:type a rdf:Property .

FOAF
Friend of a Friend
prefix foaf <http://xmlns.com/foaf/0.1/>

Important Elements
foaf:Person
foaf:name
foaf:mbox
foaf:knows

vu:Rinke_Hoekstra rdf:type foaf:Person ;
foaf:knows vu:Albert_Merono ;
foaf:name "Rinke Hoekstra" .

DC/DCT
Dublin Core / Dublin Core Terms
prefix dct <http://purl.org/dc/terms>

Important Elements
dct:creator
dct:created
dct:description
dct:title

vu:SWPrimer dct:creator vu:Rinke_Hoekstra ;
dct:created "2012"^^xsd:date;
dct:title "A Semantic Web ..." .

SKOS
Simple Knowledge Organization System
prefix skos <http://www.w3.org/2004/02/skos/core#>

Important Elements
skos:exactMatch

Good for using things that are *almost* the same. This is an informal way of establishing sameness.

```
fb:594486635    fb:likes    fb:105638652803531 .
fb:105638652803531  skos:exactMatch  imdb:tt0120669 .
```

For common prefixes and their namespaces:
http://prefix.cc

Social Side of Semantics

Berners-Lee: Using shared vocabularies is the key to establishing consistent semantics.

URI's may not always be unique. They are not a global identification service, but simply strings.

Rule of Thumb: "Only create vocabularies/URIs in a namespace within your own domain." (Important for trust).

Example in Action: Linking Vocabularies

Turtle File:

```
@prefix fb: <https://graph.facebook.com/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

fb:594486635      fb:likes      fb:289482145973 , fb:109327925759703.

fb:109327925759703  fb:name      "Waking Life"@en ;
                    fb:category  fb:movie .

fb:32985985640    fb:name      "Zeitgeist"@en ;
                  fb:category  fb:community .

fb:movie          rdfs:label   "Movie"@en .
fb:community      rdfs:label   "Community"@en .

fb:community      rdfs:subClassOf    fb:movie .
fb:category       rdfs:subPropertyOf rdf:type .
```

This is Facebook's way of defining a 'type' relationship. It is used instead of 'rdf:type'.

This is triple aligns Facebook and RDF schema by bringing Facebook's 'category' property in line with 'type' property of of RDF.

Without this triple, results of the example SPARQL would have been empty, as there would not be any 'rdf:type' properties in the Turtle file. However, now that 'fb:category' is related to 'rdf:type', asking for 'rdf:type' in SPARQL also returns all categories.

SPARQL Query:

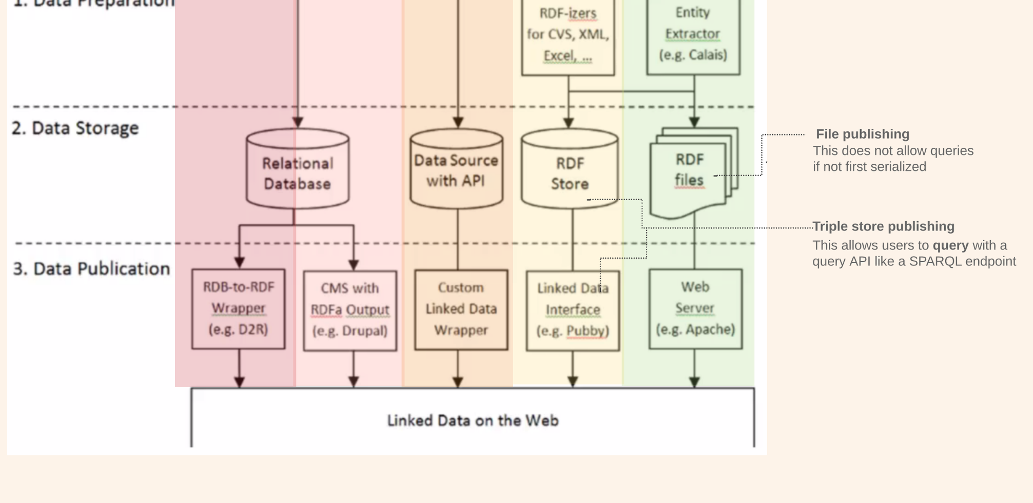
```
PREFIX fb: <https://graph.facebook.com/>

SELECT ?title WHERE {
    ?movie a      fb:movie ;
           fb:name ?title .
}
```

Query Result:

?title
"Waking Life"@en
"Zeitgeist"@en

USAGE

Accessing and Publishing RDF & Linked Data



File publishing
This does not allow queries if not first serialized

Triple store publishing
This allows users to **query** with a query API like a SPARQL endpoint

TURTLE SYNTAX

NAMESPACES AND PREFIXES

```
@prefix dbpedia:  <http://dbpedia.org/resource/> .
@prefix :  <http://dbpedia.org/ontology/> .
```

Default namespace

Qualified name, Prefixed namespace

'rdf:' is a shorthand for: http://xmlns.com/foaf/0.1/

'a' is shorthand for 'rdf:type:'

```
dbpedia:Amsterdam a :Place .
```

Non-abbreviated version:

```
<http://dbpedia.org/resource/Amsterdam rdf:type: <http://dbpedia.org/ontology/Place> .
```

SHORTHANDS

Default Abbreviations

```
dbpedia:Amsterdam rdf:type: :Place .
```

Shorthand

```
dbpedia:Amsterdam a :Place .
```

Subject Sharing

```
dbpedia:Amsterdam :officialName "Amsterdam" .
dbpedia:Amsterdam :areaTotal 219320000 .
dbpedia:Amsterdam :isCapital true .
```

Subject abbreviation with ';'

```
dbpedia:Amsterdam :officialName "Amsterdam" ;
                  :areaTotal 219320000 ;
                  :isCapital true .
```

Subject and Object Sharing

```
dbpedia:Netherlands :rdfs:label "Nederland"@nl .
dbpedia:Netherlands :rdfs:label "The Netherlands"@en .
dbpedia:Netherlands :rdfs:label "Pays-Bas"@fr .
```

Subject and predicate abbreviation with ','

```
dbpedia:Netherlands :rdfs:label "Nederland"@nl ,
                                "The Netherlands"@en ,
                                "Pays-Bas"@fr .
```

Subject Sharing and Subject & Object Sharing Combined

```
dbpedia:Netherlands :rdfs:label "Nederland"@nl ,
                                "The Netherlands"@en ,
                                "Pays-Bas"@fr ;
                    :capital    dbpedia:Amsterdam .
```

LITERALS

```
dbpedia:Amsterdam :officialName "Amsterdam" .
dbpedia:Amsterdam :areaTotal 219320000 .
dbpedia:Amsterdam :isCapital true .
```

String

Number (integer or floats can be written this way)

Boolean

Formal way to write literals:

```
dbpedia:Amsterdam :areaTotal "219320000"^^xsd:integer .
```

Literals are written in quotes

Literal type specifier

TAGS

```
dbpedia:Amsterdam rdfs:label "Amsterdam"@nl .
```

Specifies language

BLANK NODES

Blank Node: An object that has no URI, and is used in a similar fashion to the connector element of UML schemas. It serves as a terminal or intermediary between two or more elements.

Normal Syntax

```
dbpedia:VU :address
        [
            :place dbpedia:Amsterdam ;
            :street "De Boelelaan" ;
            :number "1081" ;
            :postcode "1081 HV"
        ] .
```

Think of it as passing this property directly to the next ones. As a property jumping point.

Formal Syntax

```
dbpedia:VU :address _:blankAddressNode@nl .

_:blankAddressNode :place dbpedia:Amsterdam ;
                   :street "De Boelelaan" ;
                   :number "1081" ;
                   :postcode "1081 HV" .
```

This name is a unique name *only* within the RDF file it is currently in; this is not a universally recognized blank node.

'_:' denotes blank node
':' denotes default namespace, which was previously set to <http://dbpedia.org/ontology/> .

Collections

```
:subject :predicate ( :a :b :c ) .
```

This is a blank node that houses a *collection*.

Breaking Down a SPARQL Query

'Prefix' shortcuts (optional)

Prefixes in SPARQL:
@PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@PREFIX dbpedia: <http://dbpedia.org/resource/>.
@PREFIX dbpedia-owl: <http://dbped ia.org/ontology/>.

Qualified name; Namespace prefix — URI

Keyword — Query

Result clause: SELECT ?location
"Select/return ?locations which..."

Dataset (optional): FROM example:graph1

WHERE{

Subject — Property/Predicate — Object

swp:BaronWayBuilding dbpedia-owl:location dbpedia:Amsterdam@en .
"...include a 'BaronWayBuilding located in Amsterdam' ..."

swp:BaronWayBuilding dbpedia-owl:location ?location .
"and for all triples that satisfy the above condition, return their ?location"

FILTER(filter_query_here)

}

LIMIT 10
"Limit number of returned patterns to 10." (Useful for large triple stores [i.e., databases])

ORDER BY ASC(?location)
"Order results alphabetically."

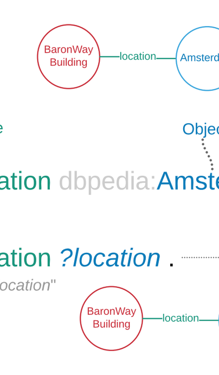
Triple pattern: A triple with one or more variables (i.e., instead of values)

Query pattern

Graph pattern: An RDF graph that has some nodes or edges as variables. It consists of multiple triple patterns.

Filters

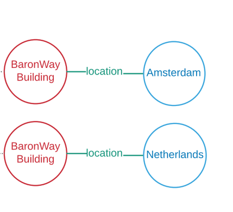
Query modifiers (optional)



Tag: Provides further information about literals (in this example, language that the string is written in)

Returns

?location
dbpedia:Amsterdam
dbpedia:Netherlands



Bindings (i.e., a table is returned, and not triples)

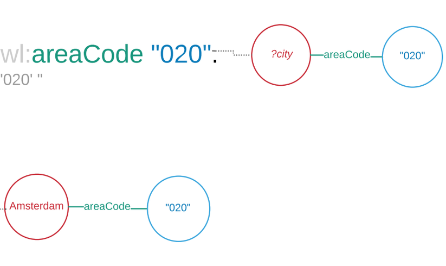
A Simpler Example

SELECT(?city)
"Select/return ?cities where..."

WHERE{
?city dbpedia-owl:areaCode "020".
"...the city's area code is '020' "
}

Returns

?city
Amsterdam



Shorthands

Comma: Add new object while reusing the statement up to this point (i.e., reuse the subject and predicate).

```
ex:ourHouse  rdf:type       ex:House, ex:ourThing ;
             ex:locatedIn   ex:ourForest .
```

Semicolon: Add new predicate and object while reusing the statement up to this point (i.e., reuse the subject).

Multiple Queries with SELECT

SELECT(?predicate ?object)
WHERE{
swp:BaronWayBuilding ?predicate ?object .
"Select all triples that has 'BaronWayBuilding' as subject."
"Return all information attached to "BaronWayBuilding".
}

Returns

?predicate	?object
swp:hasNumberOfBedrooms swp:isPartOf	3 swp:BaronWayBuilding

Union = Conjunction

Property Paths (Chain Queries)

A property path uses a 'pipeline' between statements instead of using UNION keyword, in order to retrieve multiple elements related to subject. Essentially, it is the act of walking over the same property relation more than once using statements that transition into each other. Using them is optional (i.e., same things can be achieved by using keywords such as UNION; yet, when they are used, property paths make code more concise and clear.)

Example #1

SELECT(?person ?givenName)
WHERE{
?person vcard:familyName "Smith" .
"Select all persons with family name "Smith"...
After the first line is ran, the results are kept in memory, and the second line is ran on this limited set that is in memory.
?person vcard:givenName ?givenName .
"...And then, from this selection, select a subgroup of all persons with a 'givenName' "
}
"...i.e., return all persons with the family name smith, and also return their given names."

Returns

?person	?givenName
sn:employee1 sn:employee2	Heidi John

Example #2

SELECT(?apartments)
WHERE{
?apartments swp:isPartOf ?buildings.
"Return all apartments that are part of ?building..."
?buildings dbpedia-owl:location dbpedia:Amsterdam.
"...?buildings being all buildings located in Amsterdam."
}
"...i.e., return all apartments (of buildings) located in Amsterdam."

The same statement in concise notation

?apartments swp:isPartOf / dbpedia-owl:location dbpedia:Amsterdam.

List of Property Path Operators

Syntax Form	Property Path Expression Name	Matches
iri	PredicatePath	An IRI. A path of length one.
^elt	InversePath	Inverse path (object to subject).
elt1 / elt2	SequencePath	A sequence path of elt1 followed by elt2.
elt1 | elt2	AlternativePath	A alternative path of elt1 or elt2 (all possibilities are tried).
elt*	ZeroOrMorePath	A path that connects the subject and object of the path by zero or more matches of elt.
elt+	OneOrMorePath	A path that connects the subject and object of the path by one or more matches of elt.
elt?	ZeroOrOnePath	A path that connects the subject and object of the path by zero or one matches of elt.
!iri or !(iri1| ...|irin)	NegatedPropertySet	Negated property set. An IRI which is not one of iri1. !iri is short for !(iri).
!^iri or !(^iri1| ...|^irin)	NegatedPropertySet	Negated property set where the excluded matches are based on reversed path. That is, not one of iri1...irin as reverse paths. !^iri is short for !(^iri).
!(iri1| ...|irij|^irij+1| ...|^irin)	NegatedPropertySet	A combination of forward and reverse properties in a negated property set.
(elt)		A group path elt, brackets control precedence.

Multiple Queries with UNION and OPTIONAL

Example #1

SELECT(?apartments ?labels)
WHERE{
?apartments dbpedia-owl:location dbpedia:Amsterdam.
"Select all apartments that are located in Amsterdam (which is specified by "location" property)..."
UNION
"...and add to this selection..." Think of this like a '+' sign between the lines above and below it.
?apartments dbpedia-owl:locationCity dbpedia:Amsterdam.
"...all apartments that are located in Amsterdam (which is specified by "locationCity" property)..."
OPTIONAL
"...and if possible, also..."
?apartments rdfs:label ?labels.
"...return the human readable labels of all these apartments that are now selected. (And if there are no labels, just don't return it for that binding.)"
}

Returns

?apartments	?labels
swp:BaronWayApartment swp:FloridaAveStudio	swp:Baron Way Apartment for Rent

Can also be written as:
?apartments dbpedia-owl:location | dbpedia-owl:locationCity dbpedia:Amsterdam.

Example #2

Query:

```
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
PREFIX sn: <http://www.snee.com/hr/>

SELECT ?givenName ?familyName ?oDate
WHERE
{
  ?person vcard:given-name ?givenName .
  ?person vcard:family-name ?familyName .
  ?person sn:completedOrientation ?oDate .
}
```

Result:

?givenName	?familyName	?oDate
John	Smith	2015-01-30
John	Smith	2015-03-15
Heidi	Smith	2015-01-30

John appears twice because he has two orientation dates

Missing results beause some subjects do not have an 'odate' (i.e.,orientation date')

Query:

```
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
PREFIX sn: <http://www.snee.com/hr/>

SELECT ?givenName ?familyName ?oDate
WHERE
{
  ?person vcard:given-name ?givenName .
  ?person vcard:family-name ?familyName .
  OPTIONAL { ?person sn:completedOrientation ?oDate . }
}
```

Result:

?givenName	?familyName	?oDate
John	Smith	2015-01-30
John	Smith	2015-03-15
Heidi	Smith	2015-01-30
Jane	Berger	
Francis	Jones	

OPTIONAL keyword allows subjects witout dates to also be included in the results set.

Example #3: Junction and Disjunctions

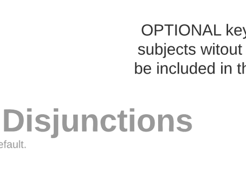
Search scope narrows with each query

And (Conjunction) — This is what is retrieved by default.

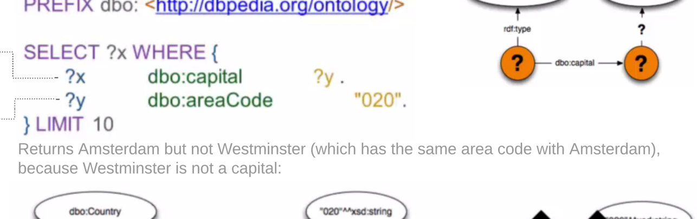
PREFIX dbo: <http://dbpedia.org/ontology/>

1. Run this query and store results
2. Run this query on the results of the previous query
3. Return the result

SELECT ?x WHERE {
?x dbo:capital ?y .
?y dbo:areaCode "020".
} LIMIT 10



Returns Amsterdam but not Westminster (which has the same area code with Amsterdam), because Westminster is not a capital:



Or (Disjunction) with UNION

Search scope does not narrow with each query

PREFIX dbo: <http://dbpedia.org/ontology/>

1. Run this query and store results
2. Run this query independently from the first one and store results
3. Combine results and return them

SELECT ?y WHERE {
{ ?y dbo:areaCode "020". }
UNION
{ ?y dbo:areaCode "010". }
} LIMIT 10

Returns Amsterdam and Rotterdam, because they are the cities that match the queried area codes (since Westminster has the same area code with Amsterdam, it should have also been returned.):

Or (Disjunction) with UNION and OPTIONAL

PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?x ?y WHERE {
?y dbo:areaCode "020".
OPTIONAL { ?x dbo:capital ?y . }
} LIMIT 10

1. Run this query and store results
2. Run this query on the results of the previous query
3. Return the result

Returns Amsterdam and Westminster (area code match), and also Netherlands:

Search scope narrows for the optional query with each line, but remains the same for main query (unless the next query is also a main query that is a junction [i.e., 'and'] clause).

Filters: Literals

Example #1

SELECT(?apartments)
WHERE{
?apartments swp:hasNumberofBedrooms ?bedroomNumbers.
"Return the number of bedrooms of all apartments and keep this in memory..."
FILTER (?bedroomNumbers > 2).
"...and then filter the results to return only those have more than 2 bedrooms."
}
"...i.e., return all apartments that has more than two bedrooms."

Returns

?apartments
swp:BaronWayApartment

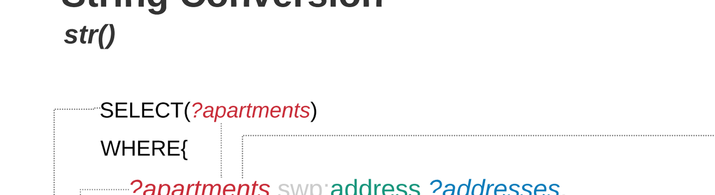
Example #2

PREFIX dbo: <http://dbpedia.org/ontology/>

SELECT ?y WHERE {
?y dbo:areaCode "020".
?y dbo:populationTotal ?x .
FILTER (?x > 500000)
}

Filters out

Returns Amsterdam but not Borehamwood, because Borehamwood's population is less than 500000:



Filters: Regular Expressions

SELECT(?apartments)
WHERE{
?apartments swp:address ?addresses.
"Return all apartments that has ?addresses..."
FILTER regex(?addresses, "^4 Baron Way").
"...?addresses being strings starting with "4 Baron Way".
}
"...i.e., return all apartments whose address starts with '4 Baron Way.' "

Returns

?apartments
swp:BaronWayApartment

Filters: Advanced Uses and Literal to String Conversion

str()

SELECT(?apartments)
WHERE{
?apartments swp:address ?addresses.
"Return all apartments that has ?addresses..."
FILTER regex(str(?apartments), "Baron").
"...?addresses being all triples whose full address contains the string "Baron".
}
"...i.e., return all apartments whose URL contains the string 'Baron' "

Returns

?apartments
swp:BaronWayApartment

<http://www.semanticwebprimer.org/ontology/apartments.ttl#BaronWayBuilding>
<http://www.semanticwebprimer.org/ontology/apartments.ttl#FloridaAveStudio>
<http://www.semanticwebprimer.org/ontology/apartments.ttl#SomeOtherBuilding>

Filters: NOT EXISTS

```
PREFIX vcard: <http://www.w3.org/2006/vcard/ns#>
PREFIX sn: <http://www.snee.com/hr/>

SELECT ?givenName ?familyName
WHERE
{
  ?person vcard:given-name ?givenName .
  ?person vcard:family-name ?familyName .
  NOT EXISTS { ?person sn:completedOrientation ?oDate . }
}
```

?givenName	?familyName
Jane	Berger
Francis	Jones

Returns subjects that do not have an oDate (i.e., Orientation date)

Advanced Uses: Variable concatenation, renaming, and long statements

BIND(), concat(), ... AS ....

```
SELECT ?givenName ?familyName ?fullName
WHERE
{
  ?person vcard:given-name ?givenName .
  ?person vcard:family-name ?familyName .
  BIND(concat(?givenName," ",?familyName) AS ?fullName).
}
```

?givenName	?familyName	?fullName
John	Smith	John Smith
Heidi	Smith	Heidi Smith
Jane	Berger	Jane Berger
Francis	Jones	Francis Jones

Concatenates '?givenName' and '?familyName' variables, and assigns the concatenated version to a new variable '?fullName'.

Creates new values via combining old ones

Renames a variable

Combines statements into longer ones.

Formatting Results: Sorting

ORDER BY DESC() | ORDER BY ASC()

SELECT (?apartments ?bedroomNumbers)
"Return apartments and number of bedrooms."
WHERE{
?apartments swp:hasNumberOfBedrooms ?bedroomNumbers.
}
ORDER BY ASC(?bedroomNumbers) — Also exists — **GROUP BY**
'ASC' for ascending, and 'DESC' for descending

?apartments	?bedroomNumbers
swp:BaronWayApartment swp:FloridaAveStudio	3 1

Formatting Results: Summarizing

COUNT
SUM
MIN
MAX

Other aggregation functions

This name has to be changed into this when displaying results (or column label would be inaccurate)

"Change variable name"

SELECT (**AVG**(?bedroomNumbers) **AS** ?averageBedroomNumbers)
WHERE{
?apartments swp:hasNumberOfBedrooms ?bedroomNumbers.
}

LIMIT 10
"Limit number of returned patterns to 10". (Useful for large triple stores [i.e., databases])

Returns

?avgNumBedrooms
2

Query Type: ASK

ASK (?apartments)
WHERE{
?apartments swp:hasNumberOfBedrooms 3.
}

Returns

'True' or 'False'

Query Type: CONSTRUCT

CONSTRUCT (?apartments swp:hasNumberOfBedrooms ?bedroomNumbers
?apartments swp:isBigApartment true)
WHERE{
?apartments swp:hasNumberOfBedrooms ?bedroomNumbers.
}
FILTER(?bedroomNumbers > 2).

A graph instead of bindings. This is effectively a subgraph of the main graph.

Returns

swp:BaronWayApartment swp:hasNumberOfBedrooms 3.
swp:BaronWayApartment swp:isBigApartment true.

Query Type: SELECT DISTINCT

Prevents results (triples?, elements of triples?) from returning more than once.

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT DISTINCT ?name WHERE {
?country rdf:type dbo:Country ;
dbo:capital ?ams .
?ams dbo:country dbpedia:Netherlands ;
rdfs:label ?name .
FILTER (lang(?name) = "en")
}

Query Type: DESCRIBE

Returns all **predicates** and **objects** about a the queried subject.

PREFIX dbr: <http://dbpedia.org/resource/>

DESCRIBE dbr:Amsterdam

Update: INSERT and INSERT DATA

Example #1: INSERT DATA

INSERT DATA
{
swp:LuxuryApartment rdfs:subClassOf swp:Apartment.
}

Example #2: INSERT

PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX ex: <http://example.com/vocab/>

INSERT {
?y ex:kengetal "020";
} WHERE {
?y dbo:areaCode "020";

Update: LOAD

LOAD <http://www.example.com/apartment.rdf>.

A graph (i.e., set of triples; triple dataset), instead of a (few) triple(s) that may be added with INSERT.

Update: DELETE (WHERE)

DELETE DATA
{
swp:LuxuryApartment rdfs:subClassOf swp:Apartment.
}

DELETE WHERE
{
?apartments swp:hasNumberOfBedrooms ?bedroomNumbers.
FILTER (?bedroomNumbers > 2).
}

The FILTER keyword is inside the brackets of WHERE this time.

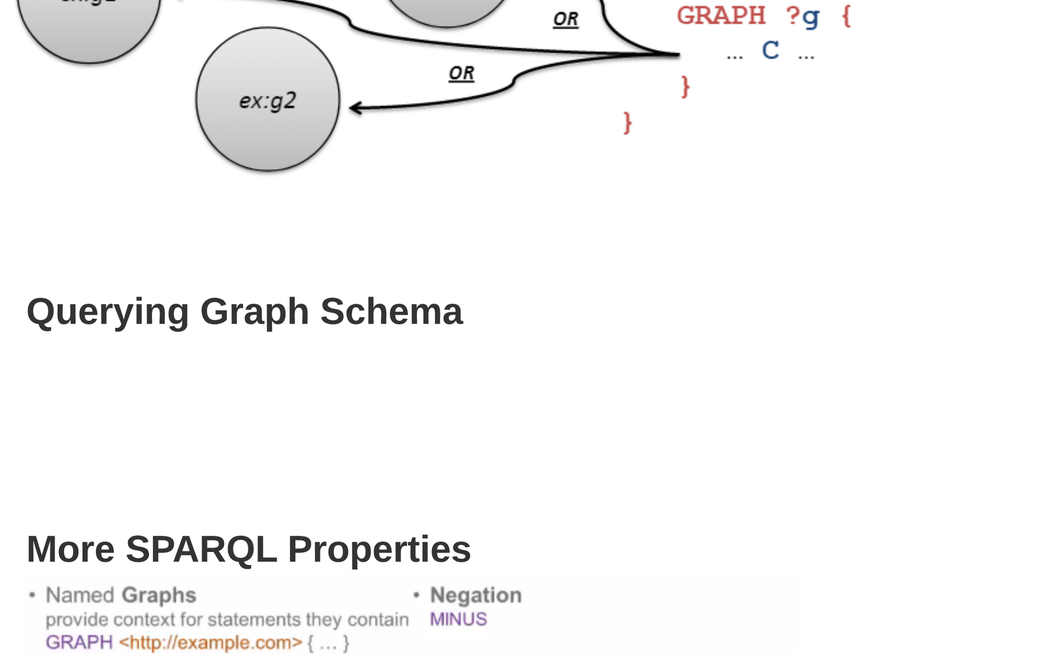
Update: CLEAR ALL

CLEAR ALL

There is no more to it. CLEAR ALL removes all content in a triple store, effectively emptying it.

FROM and FROM NAMED

A SPARQL queries a *default graph* (normally) and zero or more *named graphs* (when inside a **GRAPH** clause).



Querying Graph Schema

More SPARQL Properties

- **Named Graphs** provide context for statements they contain
  GRAPH <http://example.com> { ... }
- **Aggregates**
  COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT and SAMPLE
- **Grouping** and group **aggregates**
  GROUP BY and HAVING
- **Provide data** inline
  VALUES
- **Negation**
  MINUS
- **Functions**
  IF, BOUND, EXISTS, NOT EXISTS, IN, NOT IN
  &&, ||, =, <, >, =>, =<
  isIRI, isBlank, isLiteral, isNumeric, str, lang, datatype
  STRLEN, SUBSTR, CONTAINS, CONCAT, REGEX
  now, year, month, day, hours, minutes, seconds
  ... and more

SPARQL Cheatsheet

http://www.iro.umontreal.ca/~lapalme/ift6281/sparql-1_1-cheat-sheet.pdf