rdf:_1 rdf:type rdf:Property .
rdf:_2 rdf:type rdf:Property .
rdf:first rdf:type rdf:Property .
rdf:nil rdf:type rdf:List .
rdf:object rdf:type rdf:Property .
rdf:predicate rdf:type rdf:Property .
rdf:rest rdf:type rdf:Property .
rdf:subject rdf:type rdf:Property .
rdf:type rdf:type rdf:Property .
rdf:value rdf:type rdf:Property .

rdf:Alt rdfs:subClassOf rdfs:Container .
rdf:Bag rdfs:subClassOf rdfs:Container .
rdf:Seq rdfs:subClassOf rdfs:Container .
rdf:_1 rdf:type rdfs:ContainerMembershipProperty .
rdf:_1 rdfs:domain rdfs:Resource .
rdf:_1 rdfs:range rdfs:Resource .
rdf:_2 rdf:type rdfs:ContainerMembershipProperty .
rdf:_2 rdfs:domain rdfs:Resource .
rdf:_2 rdfs:range rdfs:Resource .
rdf:first rdfs:domain rdf:List .
rdf:first rdfs:range rdfs:Resource .
rdf:object rdfs:domain rdf:Statement .
rdf:object rdfs:range rdfs:Resource .
rdf:predicate rdfs:domain rdf:Statement .
rdf:predicate rdfs:range rdfs:Resource .
rdf:rest rdfs:domain rdf:List .
rdf:rest rdfs:range rdf:List .
rdf:subject rdfs:domain rdf:Statement .
rdf:subject rdfs:range rdfs:Resource .
rdf:type rdfs:domain rdfs:Resource .
rdf:type rdfs:range rdfs:Class .
rdf:value rdfs:domain rdfs:Resource .
rdf:value rdfs:range rdfs:Resource .
rdfs:ContainerMembershipProperty rdfs:subClassOf rdf:Property .
rdfs:Datatype rdfs:subClassOf rdfs:Class .
rdfs:comment rdfs:domain rdfs:Resource .
rdfs:comment rdfs:range rdfs:Literal .
rdfs:domain rdfs:domain rdf:Property .
rdfs:domain rdfs:range rdfs:Class .
rdfs:isDefinedBy rdfs:domain rdfs:Resource .
rdfs:isDefinedBy rdfs:range rdfs:Resource .
rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .
rdfs:label rdfs:domain rdfs:Resource .
rdfs:label rdfs:range rdfs:Literal .
rdfs:member rdfs:domain rdfs:Resource .
rdfs:member rdfs:range rdfs:Resource .
rdfs:range rdfs:domain rdf:Property .
rdfs:range rdfs:range rdfs:Class .
rdfs:seeAlso rdfs:domain rdfs:Resource .
rdfs:seeAlso rdfs:range rdfs:Resource .
rdfs:subClassOf rdfs:domain rdfs:Class .
rdfs:subClassOf rdfs:range rdfs:Class .
rdfs:subPropertyOf rdfs:domain rdf:Property .
rdfs:subPropertyOf rdfs:range rdf:Property .

rdfD1	xxx aaa "sss"^^ddd .	xxx aaa _:nnn .
	for ddd in D	_:nnn rdf:type ddd .
rdfD2	xxx aaa yyy .	aaa rdf:type rdf:Property .
rdfs1	any IRI aaa in D	aaa rdf:type rdfs:Datatype .
rdfs2	aaa rdfs:domain xxx .	yyy rdf:type xxx .
	yyy aaa zzz .	
rdfs3	aaa rdfs:range xxx .	zzz rdf:type xxx .
	yyy aaa zzz .	
rdfs4a	xxx aaa yyy .	xxx rdf:type rdfs:Resource .
rdfs4b	xxx aaa yyy.	yyy rdf:type rdfs:Resource .
rdfs5	xxx rdfs:subPropertyOf yyy .	xxx rdfs:subPropertyOf zzz .
	yyy rdfs:subPropertyOf zzz .	
rdfs6	xxx rdf:type rdf:Property .	xxx rdfs:subPropertyOf xxx .
rdfs7	aaa rdfs:subPropertyOf bbb .	xxx bbb yyy .
	xxx aaa yyy .	
rdfs8	xxx rdf:type rdfs:Class .	xxx rdfs:subClassOf rdfs:Resource .
rdfs9	xxx rdfs:subClassOf yyy .	zzz rdf:type yyy .
	zzz rdf:type xxx .	
rdfs10	xxx rdf:type rdfs:Class .	xxx rdfs:subClassOf xxx .
rdfs11	xxx rdfs:subClassOf yyy .	xxx rdfs:subClassOf zzz .
	yyy rdfs:subClassOf zzz .	
rdfs12	xxx rdf:type rdfs:ContainerMembershipProperty .	xxx rdfs:subPropertyOf rdfs:member .
rdfs13	xxx rdf:type rdfs:Datatype .	xxx rdfs:subClassOf rdfs:Literal .

@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix ex: <http://example.org/data/> .
@prefix exv: <http://example.org/vocab#> .

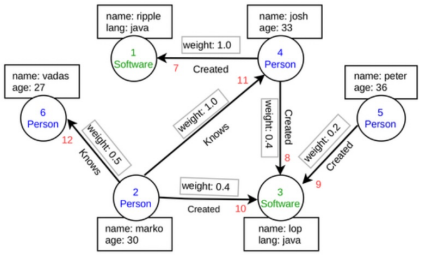
ex:EID15 rdf:type exv:OpenMarket, exv:FoodFestival, exv:DrinksFestival ;
  exv:name "Ñam" ;
  exv:start "2018-03-22T12:00:00"^^xsd:dateTime ;
  exv:end "2018-03-29T20:00:00"^^xsd:dateTime ;
  exv:venue ex:SantaLucía .
ex:SantaLucía exv:city ex:Santiago .
ex:EID16 rdf:type exv:OpenMarket, exv:FoodFestival ;
  exv:name "Food Truck"^^xsd:string ;
  exv:venue ex:Sotomayor, ex:PiscinaOlímpica .

Directed-Edge Labeled Graph

Heterogenous Graph

Sebuah edge pada graf ini disebut homogen bila dia berada di antara dua node yang tipe atau labelnya sama. Selain itu, disebut heterogen, misalnya antara chile dan peru itu homogen, dan capital itu heterogen. Graf ini berguna untuk mempartisi node berdasarkan tipenya, berguna untuk machine learning task tertentu. Beda dengan DELG, graph ini hanya mendukung relasi many-to-one antara node dengan tipenya, (setiap node diasosiasikan dengan satu tipe aja ga bisa lebih, atau kaya interface)

Property Graph



$\mathbf{V}$ = {1,2,3,4,5,6};
$\mathbf{E}$ = {(4,7,1), (4,8,3), (5,9,3), (2,10,3), (2,11,4), (2,12,6)}
$\mathbf{P_v}$ = {name,age,lang} ; $\mathbf{P_e}$ = {weight}
$\mathbf{L_v}$ = {Software,Person} ; $\mathbf{L_e}$ = {Created,Knows}

$\boldsymbol{\mu}(\mathbf{V/E}, \mathbf{P_v/P_e}) = \textbf{"value"}$;
$\boldsymbol{\mu}(1,name)$ = "ripple";
$\boldsymbol{\mu}(1,lang)$ = "java";
$\boldsymbol{\mu}(2,name)$ = "josh" ;
$\boldsymbol{\mu}(2,age)$ = "33" ;
$\boldsymbol{\mu}(3,name)$ = "lop";
$\boldsymbol{\mu}(9,weight)$ = "0.2" ;
$\boldsymbol{\mu}(11,weight)$ = "1.0";
...

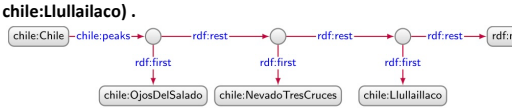
$\boldsymbol{\lambda}(\mathbf{V/E}) = \mathbf{L_v/L_e}$;
$\boldsymbol{\lambda}(1)$ = "Software";
$\boldsymbol{\lambda}(2)$ = "Person";
$\boldsymbol{\lambda}(7)$ = "Created";
$\boldsymbol{\lambda}(8)$ = "Created";
$\boldsymbol{\lambda}(11)$ = "Knows";
$\boldsymbol{\lambda}(12)$ = "Knows";
$\boldsymbol{\lambda}(5)$ = "Person";

DELG menawarkan model yang lebih minimal, sementara property graph menawarkan yang lebih flexibel.

Blank nodes have identifier, they are unique within a single RDF doc, cannot be referred to/from outside the document, rewriting or reloading the graph may change blank node identifiers

RDF list representation with blank nodes.

chile:Chile chile:peaks
(chile:OjosDelSalado chile:NevadoTresCruces
chile:Llullailaco) .



ex:Chutney ex:hasIngredient _:b1, _:b2 .
ex:Chutney ex:hasIngredient
  [ex:ingredient ex:greenMango ;
    ex:amount "1 lb."],
  [ex:ingredient ex:CayennePaper ;
    ex:amount "1 tsp."].

Diberikan sebuah BGP Q, yang di atas tadi, definisikan Var(Q) adalah himpunan semua variabel yang muncul pada Q, variabel itu yang ga konstan dan belum dibinding valuenya. Misalkan μ:Var → Con , **merupakan sebuah partial mapping**, dari variabel ke konstan pada domainnya si μ, Con nya bisa berupa V atau E yang penting constant semua isinya.

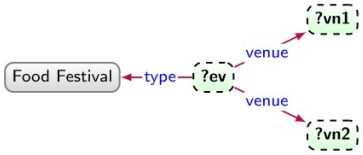
dom(μ): Semua variabel pada mapping μ,
μ dikatakan solusi terhadap G jika dan hanya jika data graphnya $\mu(Q)$ **merupakan subgraph dari data graph** G. $Q(G)$ itu merupakan multiset of solution mapping.

$$Q(G) = \{\mu \mid \mu(Q) \subseteq G \ \wedge \text{dom}(\mu) \subseteq \text{Var}(Q)\}$$

Bila kita tidak bisa menemukan solution mapping μ, yang merupakan subgraph terhadap query Q nya itu G, dan domain dari μ nya merupakan variabel dari Q, artinya semua μ itu memetakan variabel dari Q. Kenapa dom(μ) ⊆ Var(Q), karena tidak semua domainnya perlu kita jawab dan tidak semuanya perlu kita kembalikan semuanya, tapi bisa saja ada query, misalnya kaya

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX xsd: <http://www.w3.org/2001/>
PREFIX ex: <http://example.org/data/>
PREFIX exv: <http://example.org/vocab#>
select ?ev ?vn1 ?vn2
where {
?ev rdf:type exv:FoodFestival ;
exv:venue ?vn1, ?vn2 .
}

Homomorphism-based semantics will yield:



Intinya: untuk setiap connected subgraph, dimatching bentuknya apakah sama dengan query. SPARQL juga homomorphism

?ev	?vn1	?vn2
EID16	Piscina Olimpica	Sotomayor
EID16	Sotomayor	Piscina Olimpica
EID16	Piscina Olimpica	Piscina Olimpica
EID16	Sotomayor	Sotomayor
EID15	Santa Lucia	Santa Lucia

In isomorphism based semantics, the last three mappings ga.
Intinya: untuk setiap connected subgraph, dimatching bentuknya apakah sama dengan query.
- Isomorphism, dia bakal mapping untuk setiap vertex secara bijektif (kodomain sama dengan range).
- Homomorphism, mappingnya tidak harus bijektif.
- Epimorphism: surjective doang
- Monomorphism: injective doang

Setelah dimap, cek apakah edges hasil map needlenya sama dengan graph haystack. Untuk semua sifat fungsi:
- Injective (One-to-one): Boleh menjomblo, tidak boleh poligami
- Surjective (onto): Tidak boleh menjomblo, boleh poligami
- Bijektif (One to one and onto): Tidak boleh menjomblo, tidak boleh poligami

Perhatikan bahwa dalam query ini ada beberapa node yang mesti dimap ke node spesifik tertentu sehingga querynya lebih sempit. Untuk SPARQL, kita pakainya homomorphism.

Perhatikan bahwa disini yang perlu kita ambil cuma variabel ?ev nya, tidak perlu dom(μ) = Var(Q).
Kita katakan data graph G_1 itu subgraph dari G_2 jika dan hanya jika $V_1 \subseteq V_2, E_1 \subseteq E_2$ dan $L_1 \subseteq L_2$
μ = { ?ev ↦ EID16, ?vn1 ↦ Piscina Olimpica, ?vn2 ↦ Sotomayor }
$\mu(Q)$ = {(EID16, type, Food Festival), (EID16, venue, Piscina Olímpica), (EID16, venue, Sotomayor) } Is $\mu(Q) \subseteq G$? **Yes**.
μ = { ?ev ↦ EID16, ?vn1 ↦ Sotomayor, ?vn2 ↦ Piscina Olimpica }
$\mu(Q)$ = {(EID16, type, Food Festival), (EID16, venue, Sotomayor), (EID16, venue, Piscina Olimpica) Is $\mu(Q) \subseteq G$? **Yes**.
μ = { ?ev ↦ EID16, ?vn1 ↦ Piscina Olimpica, ?vn2 ↦ Piscina Olímpica }
$\mu(Q)$ = {(EID16, type, Food Festival), (EID16, venue, Piscina Olimpica) } Is $\mu(Q) \subseteq G$? **Yes**.
Perhatikan bahwa $\mu(Q)$ merupakan set atau himpunan.

Untuk ngematch literal, datatypenya juga harus cocok, bagian yang di dalam tanda petik itu namanya bentuk lexical.

ex:ex1 exv:p "test" .
ex:ex2 exv:p "test"^^xsd:string .
ex:ex3 exv:p "test"@en .
ex:ex4 exv:p "42"^^xsd:integer .
ex:ex5 exv:p 42 .

SELECT * WHERE {
?s exv:p "test" .
}

Query di atas bakal ngereturn ex1, ex2 doang, karena yang en itu typenya `rdf:langString`.

Semantic Web

- Sudut pandang konten: Lapisan metadata baru pada web yang mendeskripsikan konten dengan istilah kosa katanya dibagikan.
  - Web menjadi sebuah sistem basis data global
  - Jejaring data
- Sudut pandang aplikasi: Web yang dimengerti oleh mesin
  - Makna atau semantik dari suatu konten yang dapat diakses oleh mesin
  - Layanan web yang pintar
  - Interoperabilitas dari makna tersebut oleh berbagai sistem komputer
- Sudut pandang teknologi: sebagai sebuah lapisan di atas XML.

Contoh knowledge graph: Wikidata
- Data graph itu data model yang berbentuk graph
- Knowledge mengarah kepada sesuatu yang kita sudah ketahui. Knowledge pada KG (Knowledge Graph)
- Simple Statements
- Quantifed Statements
  - Setiap ibukota adalah sebuah kota, kita tidak mungkin membuat edge. untuk semuanya, karena ini tuh itungannya lebih ke ontologi atau aturan. Lebih ekspresif sekadar dibikin node Untuk mendapatkan lebih banyak informasi, kita bisa menyimpulakn dengan metode deduktif atau induktif
- Kita bisa menyimpulkan bahwa Jakarta itu adalah sebuah kota yang benar juga.

• Tapi di sini kita extend ideanya URL untuk menggunakan Internationalized Resource Identifier (IRI) — dulunya namanya Uniform Resource Identifier (URI).
• Dia merupakan identifier, tapi juga merupakan lokasi di web.
• Setiap URL itu IRI, karena setiap URL itu identifier ke dokumen di web.

IRI ::= scheme:[//authority]path[?query][#fragment]
• Scheme: http, https, ftp, mailto, urn **Case insensitive**
• Authority: kgbook.org, john@example.com, example.org:8080 **Case insensitive**
• Path: /etc/passwd, this/path/with/-:_ /is/../okay **Case sensitive, May be empty**
• Query: q=Semantic+Web+book **Case sensitive**, **May be empty**
• Fragment: #section1 **Case sensitive, May be empty**
Kita prefer menggunakan https atau http karena sebenarnya udah kebind sama web protocol. IRI itu harus persistent, harus tetap hidup dan menunjuk ke resource yagng sama selamanya. Bagian querynya mesti kosong.
• IRI itu globally unique, hostnamenya merujuk ke suatu data owner tertentu.
• Uniquenessnya dipastikan oleh data owner
• Jadi dia globally unique.

Syntax Form	Matches
`uri`	A URI or a prefixed name. A path of length one.
`^elt`	Inverse path (object to subject).
`(elt)`	A group path `elt`, brackets control precedence.
`elt1 / elt2`	A sequence path of `elt1`, followed by `elt2`
`elt1 ^ elt2`	Shorthand for `elt1 / ^elt2`, that is `elt1` followed by the inverse of `elt2`.
`elt1 | elt2`	A alternative path of `elt1`, or `elt2` (all possibilities are tried).
`elt*`	A path of zero or more occurrences of `elt`.
`elt+`	A path of one or more occurrences of `elt`.
`elt?`	A path of zero or one `elt`.
`elt{n,m}`	A path between n and m occurrences of `elt`.
`elt{n}`	Exactly `n` occurrences of `elt`. A fixed length path.
`elt{n,}`	`n` or more occurrences of `elt`.
`elt{,n}`	Between 0 and `n` occurrences of `elt`.

- RDF parsers: Buat baca RDF
- RDF serializers: Buat nulis ke memori
Biasanya di satu library sudah include keduanya dan bisa diserialisasi ke berbagai macam format, dari n-triples, turtle, json-ld, dan urutannya bisa berbeda, serta blank node IDnya tapi merepresentasikan graf yang sama. Aplikasinya bisa pakai Apache Jena, Apache any23, rdflib, Eclipse rdf4j, JsonLD, dkk.

RDF store itu suatu basis data yang menyimpan data RDF, beberapa ada yang merupakan ekstensi dari RDBMS biasa, dibikin jadi semacam triple table, edgesnya bisa dibuat binary relations, bisa juga ada property tables. Data pada RDF store bisa diakses dengan yang namanya SPARQL query, nanti akan diproses oleh query engine, setiap RDF store ada query enginenya, querynya dijalankan ga dari file RDF nya tapi di basis datanya, ada pula protokolnya untuk komunikasi query dan hasilnya. Ada pula reasoning engines, yang bisa memproses inferensi logika dari data-datanya, termasuk dari kata kerja (edge, verb) dan dua objek (node) nya. Istilah data federation ini: penyatuan data ini biasanya untuk setiap triple disimpan dalam satu penyimpanan. Gunanya agar querynya lebih mudah.

Salah satu contoh dari RDF ini ialah pada schema.org. Jika kita buat menggunakan schema.org, bisa meningkatkan expose dari SEO ke website kita.
Syntax vs. Semantics
Syntax
- Merupakan Proportitional Variables
- Syntax merupakan koleksi simbol dan term bersamaan dengan rules, untuk membentuk suatu sentence yang dianggap valid.
- Koleksi dari terms itu kita sebut dengan vocabulary.

Statement
- Setiap proporsitional variable itu adalah sebuah statement.
- Jika misalnya a dan b adalah statement, maka:
  - $a \wedge b$ merupakan statement
  - $a \vee b$ merupakan statement
  - $a \Longrightarrow b$ merupakan statement
  - $a \Leftrightarrow b$ merupakan statement

Semantics (Basis logic)
- Merupakan himpunan semesta.
- mapping statement to himpunan semesta
- Meaning dari syntax.
- Didefinisikan dengan himpunan,
- Interpretasi yang menspesifikan setiap simbol dan statement itu artinya apa dan kapan sebuah statement disebut true atau false.

Syntax is the grammatical structure of the text, whereas semantics is the meaning being conveyed
Entailment
Hubungan antara statement yang true saat diketahui suatu himpunan statement itu true.
Suatu himpunan G disebut **entails** sebuah himpunan lain H, bila setiap interpretasi I dari G yang membuat G benar juga membuat H benar.

$$G: = \{ p \Rightarrow q, p \}$$
$$H: \{q\}$$

Jadi cara untuk mengecek entailment adalah looping setiap interpretasi yang membuat G benar, terus apabila ada satu saja yang membuat H salah, maka itu tidak entail.
Reasoning/inference:
Diberikan dua buah himpunan G dan H, cek apakah G entails H, intinya ini proses inferensi.
Semantic dari RDF Graph
- Graf RDF itu hanya himpunan triple
  - Tidak ada deskripsi tentang struktur dan bentuknya
  - Tidak ada makna spesifik yang diberikan kepada setiap IRI

Sebuah RDF graph true bila semua triplenya true, dan definisi dari triple true jika dan hanya jika ada sebuah relasi binary yang diidentifikasi dengan p yang menhubungkan entity s dan entity lain o.
- Kebanyakan triple hampir selalu true
- Jika o literal, maka bentuk lexicalnya mesti cocok dengan tipe datanya, "test"^^xsd:integer itu ill typed dan bermasalah.

RDF triple itu dinyatakan dalam bentuk (s, p, o).
- s, subject, bisaa IRI atau blank node.
- p, mesti IRI
- o, bisa IRI, blank, atau literal.

Kalau RDF graph itu himpunan triple. Dari DELG, kita tinggal menentukan node mana ayng literal dan tipenya apa, terus sisanya bisa kita IRI dan jadikan dia object entitas sendiri.

Entailment of RDF Graph
xxx ppp yyy . entails xxx ppp yyy .
xxx ppp yyy . entails xxx ppp _:nnn .
xxx ppp yyy . entails _:nnn ppp yyy .
xxx ppp "aaa"^^ddd . entails xxx ppp _:nnn . _:nnn rdf:type ddd .

Prinsip Linked Data:
1. Menggunakan URI untuk segala hal
2. Menggunakan URI HTTP untuk mengizinkan orang orang untuk mencari namanya
3. Saat seseorang mencari sebuah URI berikan informasi yang berguna menggunakan standar RDF dan syntax-syntaxnya
4. Pada informasi yang dikembali kan, masukkan pula URI HTTP dari data lain yang terhubung sehingga bisa digunakan untuk menemukan hal lain.

Cool URIs
Simple: pendek, URInya mnemonic, terutama untuk prefixnya
Stable: Harus hidup selamanya, Hindari ekstensi implementation-specific, misalnya php, html, asp
Manageable: Bisa dikelola, Masukkan tahun juga untuk URInya, kode 303 atau moved itu dibikin subdomainnya, untuk mempermudah migrasi

URI dereferencing
303
Negosiasi konten: Mengembalikan informasi yang sesuai mengenai suatu URI menggunakan HTTP. Server yang akan menentukan apa yang dikembalikan. Strategi untuk dereferencing: 303 See Other URI, Hash URI
Masalahnya objek URInya tidak bisa dikembalikan, hanya dokumen
Hash
Metode 303 butuh dua request karena redirect
Gunakan `#` atau identifier lain
Client memotong URI sampai # (remove #Seminar)
Prosedur:
Bagus untuk deskripsi RDF yang kecil:
- Hemat HTTP Request
- Tapi client mesti filter responsnya lagi

```
SELECT ?namaArtis ?namaAlbum ?tanggal WHERE {
  ...
  OPTIONAL { ?album :name ?namaAlbum } .
  FILTER (?tanggal >= "1970-01-01"^^xsd:date && ?tanggal <
"1980-01-01"^^xsd:date)
  FILTER NOT EXISTS {
    ?album :date ?tanggalLain
    FILTER (?tanggal > ?tanggalLain)
  }
}
```

```
SELECT * WHERE {
  {
    # Mencari jumlah anggota untuk semuanya
    SELECT ?band ?namaBand (COUNT(?anggota) AS
?anggotaCount) WHERE {
      ?band a :Band .
      ?band :member ?anggota .
      ?band :name ?namaBand .
    }
    GROUP BY ?band ?namaBand
  }
  #  Filter untuk anggotanya >= 5 saja
  FILTER (?anggotaCount >= 5)
}
```

```
SELECT ?judulAlbum ?judulLagu ?namaArtis ?panjangLagu
?paraPenulis WHERE {
  ?album :artist ?artis .
  ?album :name ?judulAlbum .
  ?artis :name ?namaArtis .
  {
    # Mencari semua data lagu dengan panjang dan concat dari
penulisnya
    SELECT ?album ?lagu ?judulLagu ?panjangLagu
(GROUP_CONCAT(?namaPenulis;SEPARATOR=", ") AS
?paraPenulis) WHERE {
      ...
      ?penulis :name ?namaPenulis .
      ...
    }
    GROUP BY ?album ?lagu ?judulLagu ?panjangLagu
  }
```

```
SELECT * WHERE {
  {
    SELECT (?class as ?barang) ("class" as ?jenis)
(COUNT(?instance) AS ?hitungBarang) WHERE {
      {
        SELECT DISTINCT ?class WHERE { # SyaratClass  }
      }
      # Simbol asterisk untuk mengecek 0 atau lebih rantai
      ?subclass rdfs:subClassOf* ?class .
      ?instance a ?subclass .
    }
    GROUP by ?class
  } UNION {
    SELECT (?prop as ?barang) ("property" as ?jenis) (COUNT(*) AS
?hitungBarang) WHERE {
      # Menggunakan blank node untuk menghitung banyaknya
pasangan yang cocok
      [] ?prop []
    }
    GROUP BY ?prop
  } }
```