

SPARQL: Simple Protocol And RDF Query Language

W3C Recommendation

- SPARQL 1.0 Query (2008)
- SPARQL 1.1 Query (2013)
<https://www.w3.org/TR/sparql11-query/>
- SPARQL 1.1 Update (2013)
<https://www.w3.org/TR/sparql11-update/>

SPARQL 1.1

1. SPARQL 1.1 Overview
2. SPARQL 1.1 Query Language
3. SPARQL 1.1 Update
4. SPARQL1.1 Service Description
5. SPARQL 1.1 Federated Query
6. SPARQL 1.1 Query Results JSON Format
7. SPARQL 1.1 Query Results CSV and TSV Formats
8. SPARQL Query Results XML Format (Second Edition)
9. SPARQL 1.1 Entailment Regimes
10. SPARQL 1.1 Protocol
11. SPARQL 1.1 Graph Store HTTP Protocol

SPARQL

1. SPARQL 1.1 Overview
- 2. SPARQL 1.1 Query Language**
- 3. SPARQL 1.1 Update**
4. SPARQL1.1 Service Description
- 5. SPARQL 1.1 Federated Query**
6. SPARQL 1.1 Query Results JSON Format
7. SPARQL 1.1 Query Results CSV and TSV Formats
8. SPARQL Query Results XML Format (Second Edition)
9. SPARQL 1.1 Entailment Regimes
- 10. SPARQL 1.1 Protocol**
11. SPARQL 1.1 Graph Store HTTP Protocol

SPARQL 1.1 QUERY LANGUAGE

SPARQL 1.1 Query Language

- Syntax
- Triple Pattern
- Graph Pattern Matching
- Filter
- Query Form
- Statement
- Modifier

SPARQL Syntax

Triple Pattern

- Turtle triple syntax
- Variables

`?x a foaf:Person`

`?x foaf:name "John"`

`<http://example.org/John> foaf:name ?name`

`<http://example.org/John> ?p ?v`

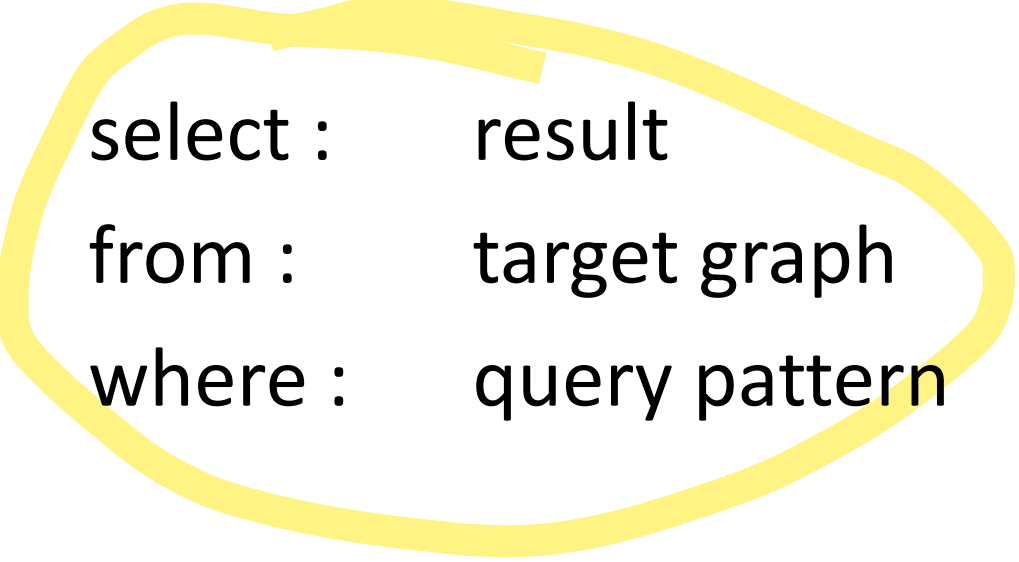
Select Query

select

from

where

Select Query



```
select :    result
from :      target graph
where :     query pattern
```

Select Query

```
select *  
where {  
    ?x rdf:type foaf:Person .  
}
```

Select Query

```
select *  
where {  
    ?x rdf:type foaf:Person ;  
    foaf:name "John" .  
}
```

Select Query

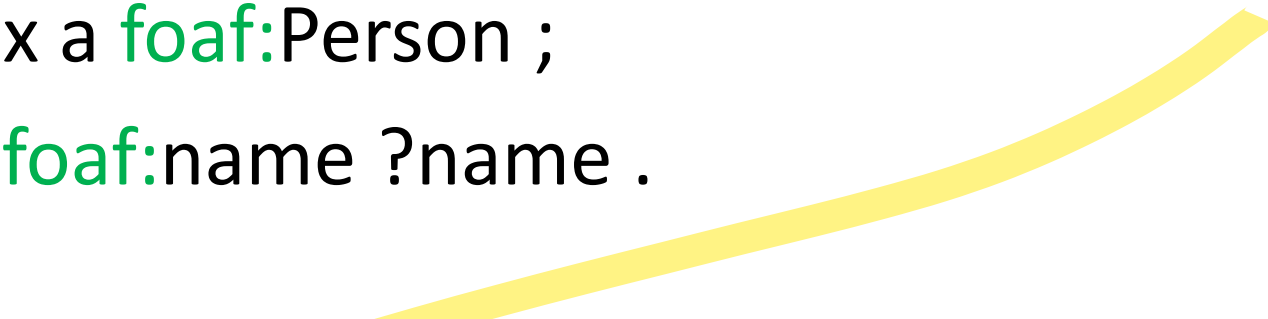
```
select *  
where {  
    ?x rdf:type foaf:Person ;  
    foaf:name "John" , ?name .  
}
```

Select Query

```
select *  
where {  
    ?x rdf:type foaf:Person ;  
    foaf:name "John", ?name .  
    ?y foaf:knows ?x  
}
```

Prefix, Namespace

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
select *
where {
  ?x a foaf:Person ;
    foaf:name ?name .
}
```



Prefix, Namespace

```
select *  
where {  
  ?x a <http://xmlns.com/foaf/0.1/Person> ;  
  <http://xmlns.com/foaf/0.1/name> ?name .  
}
```


Prefix, Namespace

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
prefix ex:   <http://example.org/ns#>
select *
where {
    ?x a foaf:Person ;
        ex:name ?name .
}
```

Literal

- XSD Datatype

"1930-01-29"^^xsd:date

"3.14"^^xsd:double

"12"^^xsd:integer

12

"true"^^xsd:boolean

true

"Never surrender"^^xsd:string

"Never surrender"

...

Literal

- Language tag

"Person"@en

"Personne"@fr

Literal

"Person"

!=

"Person"@en

<http://example.org/John>

!= "http://example.org/John"

!= "http://example.org/John"^^xsd:anyURI

Blank Node

- *Anonymous* variable
- Value of Blank Node is not returned in result

```
select *  
where {  
    _:b a foaf:Person ;  
    foaf:name ?name  
}
```

Blank Node

```
select ?x where {  
    ?x a foaf:Person ;  
    foaf:knows [ foaf:name "John" ] .  
}
```

```
select ?x where {  
    ?x a foaf:Person ;  
    foaf:knows _:b .  
    _:b foaf:name "John" .  
}
```

RDF List

- Retrieve a list with the exact number of elements

```
select * where {  
    ?x rdf:value (?y ?z)  
}
```

RDF List

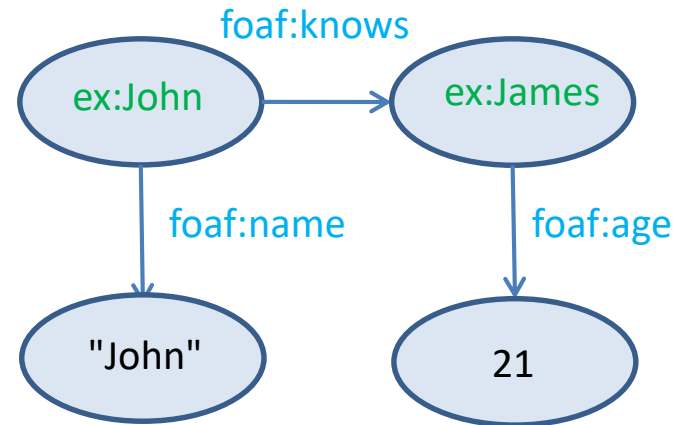
```
select * where {  
    ?x rdf:value (?y ?z)  
}
```

```
select * where {  
    ?x rdf:value [  
        rdf:first ?y ;  
        rdf:rest [ rdf:first ?z ; rdf:rest rdf:nil ]  
    ]  
}
```

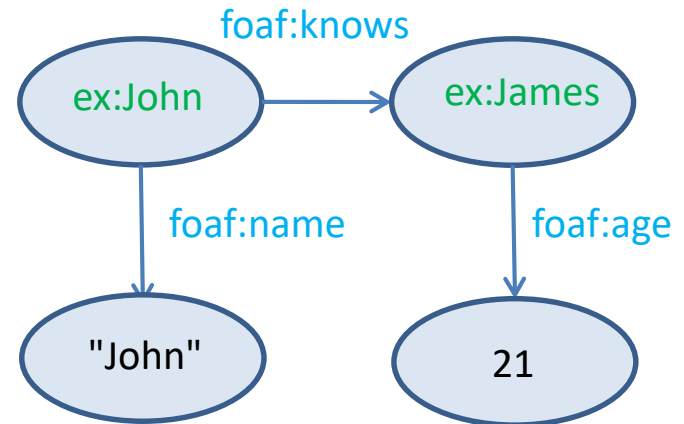
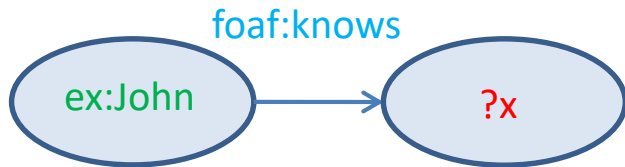

SPARQL Query Processing

- A query is a Graph Pattern (graph with variables)
- Search occurrences of Graph Pattern in RDF Graph
- Zero, one or several results

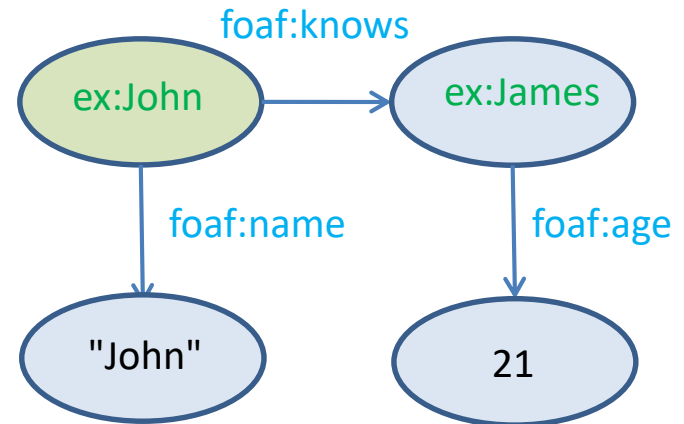
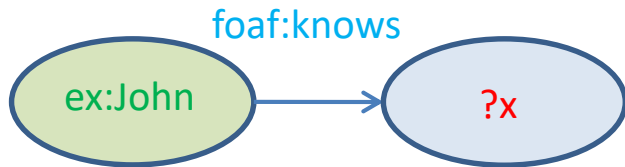
Graph Pattern Matching



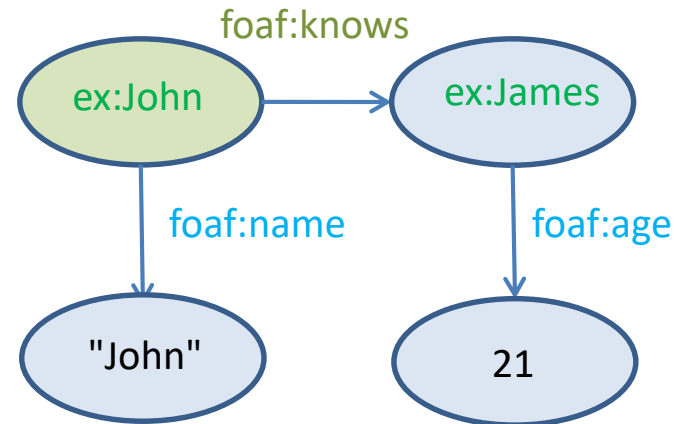
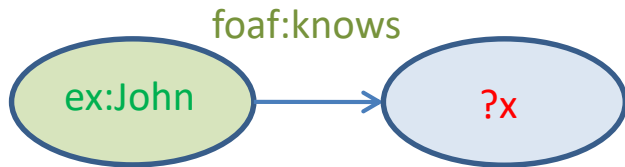
Graph Pattern Matching



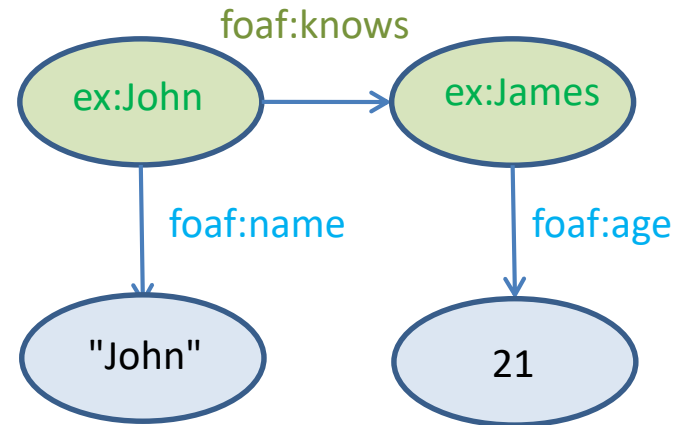
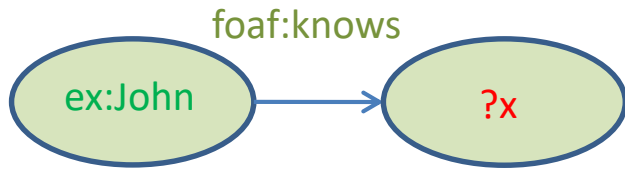
Graph Pattern Matching



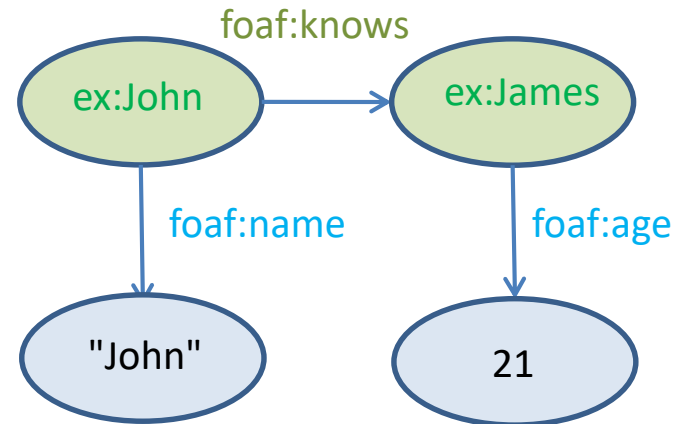
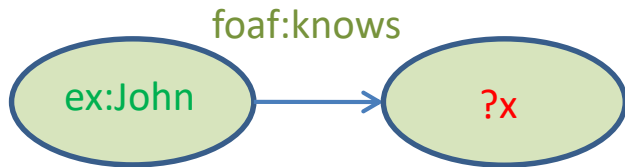
Graph Pattern Matching



Graph Pattern Matching



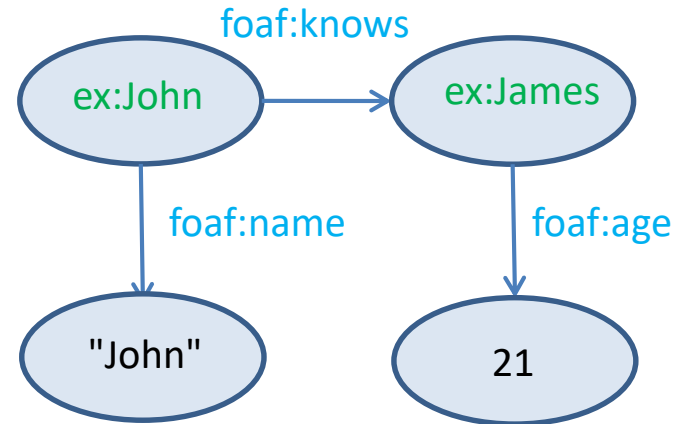
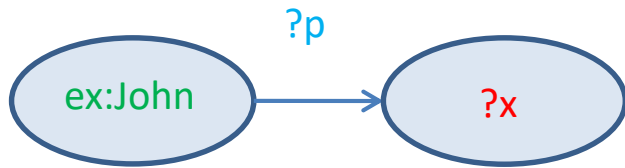
Graph Pattern Matching



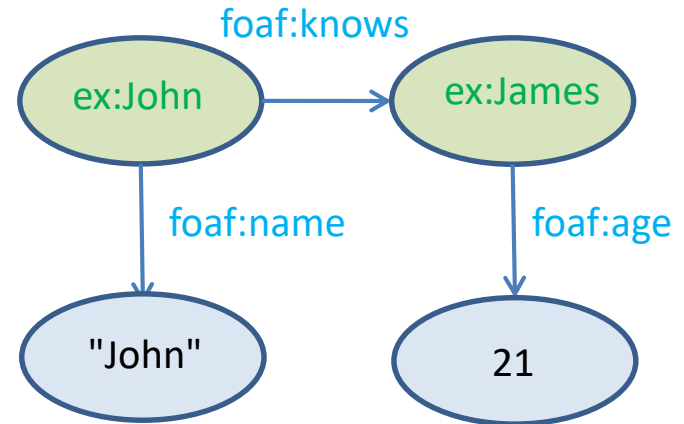
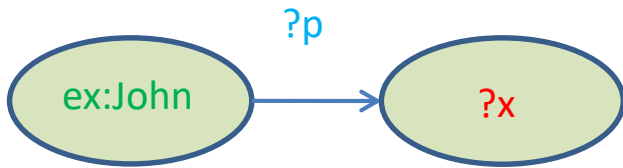
ex:John foaf:knows ?x

(1) ?x = ex:James

Graph Pattern Matching (2)



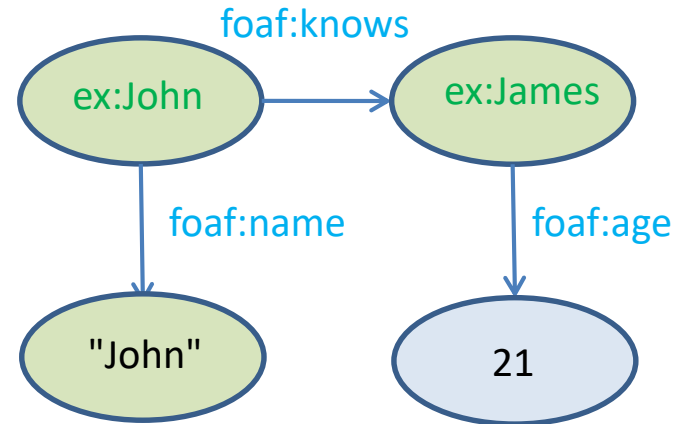
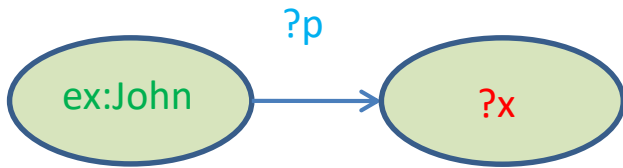
Graph Pattern Matching (2)



`ex:John ?p ?x`

(1) `?p = foaf:knows ; ?x = ex:James`

Graph Pattern Matching (2)



ex:John ?p ?x

(1) ?p = foaf:knows ; ?x = ex:James

(2) ?p = foaf:name ; ?x = "John"

SPARQL Query Processing

- Result: (multi) set of variable bindings

1. `?x = ex:John ; ?z = ex:Jack`
2. `?x = ex:Jim ; ?z = ex:Jesse`
3. `?x = ex:John ; ?z = ex:Jack`

Multiset of results

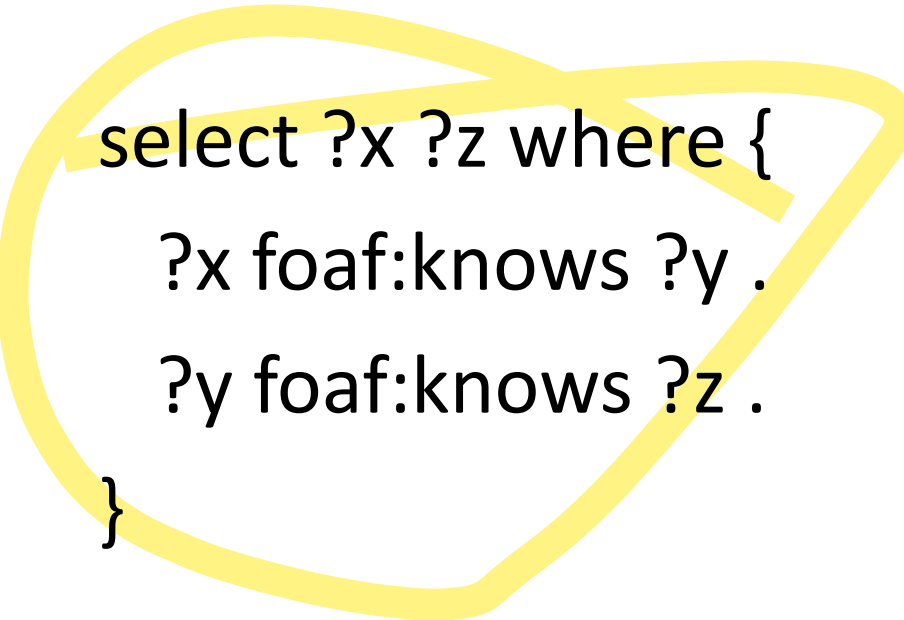
```
select ?x ?z where {  
  ?x foaf:knows ?y .  
  ?y foaf:knows ?z .  
}
```

Multiset of results

```
select ?x ?z where {  
  ?x foaf:knows ?y .  
  ?y foaf:knows ?z .  
}
```

```
ex:John foaf:knows ex:James . ex:James foaf:knows ex:Jack  
ex:John foaf:knows ex:Patty . ex:Patty foaf:knows ex:Jack
```

Multiset of results



```
select ?x ?z where {  
  ?x foaf:knows ?y .  
  ?y foaf:knows ?z .  
}
```

ex:John foaf:knows ex:James . ex:James foaf:knows **ex:Jack**
ex:John foaf:knows ex:Patty . ex:Patty foaf:knows **ex:Jack**

Multiset of results

```
select ?x ?z where {  
  ?x foaf:knows ?y .  
  ?y foaf:knows ?z .  
}
```

ex:John foaf:knows ex:James . ex:James foaf:knows ex:Jack
ex:John foaf:knows ex:Patty . ex:Patty foaf:knows ex:Jack

1. ?x = ex:John ; ?z = ex:Jack
2. ...
3. ?x = ex:John ; ?z = ex:Jack

Example

- Resources with name "Iannis" and possibly other name(s)

Example

- Resources with name "Iannis" and possibly other name(s)

`?x foaf:name "Iannis" , ?name .`

Example

- Triple with same subject and object

Example

- Triple with same subject and object

`?x ?p ?x`

Example

- Triple with property as subject

Example

- Triple with property as subject

?p ?p ?x

Filter

Filter Expression

- Reduce result multiset
- Keep results that match a condition
- Match: expression evaluates to *true*

?age >= 10

?name != "John"

isURI(?x) || isBlank(?x)

! us:boring(?course)

Filter

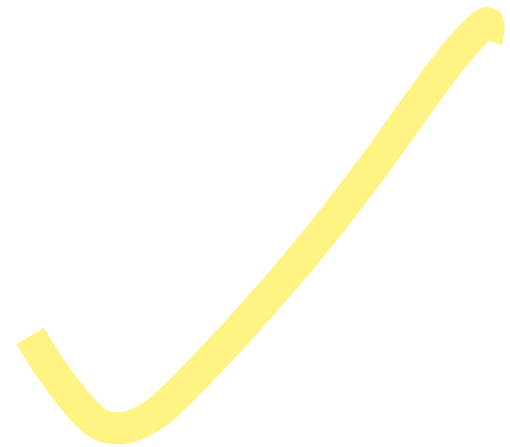
- Variable:
 1. bound by triple pattern

```
select * where {  
  ?x foaf:age ?age  
  
}
```


Filter

- Variable:
 1. bound by triple pattern
 2. tested by filter

```
select * where {  
  ?x foaf:age ?age  
  filter (?age >= 18)  
}
```



Filter

- Variable:
 - ~~1. bound by triple pattern~~
 2. tested by filter

select * where {

filter (?age >= 18)
}



Filter Language

- URI, Literal, Variable ex:John, 3.14, ?x
 - < <= >= >
 - = !=
 - ()
 - + - * /
 - && || !
 - Function
 - Exists
- ?age >= 18, ?n < "John"
- ?x != ?y, ?y = "James"
- (?n * (?n + 1))/2
- ! (?x < 0 && ?y < 0)
- datatype(?x)
- exists { ?x foaf:knows ?y }

Example

- Resources with name "Iannis" and with another name

Example

- Resources with name "Iannis" and with another name

```
?x foaf:name "Iannis" , ?name .  
filter (?name != "Iannis")
```

Example

- Resources with different values for same property

Example

- Resources with different values for same property

```
?x ?p ?y , ?z .  
filter (?y != ?z)
```

Example

- Resources with same value for different properties

Example

- Resources with same value for different properties

```
?x ?p ?v ; ?q ?v .
```

```
filter (?p != ?q)
```

Conditional Statement

- if then else

```
filter if (lang(?name) = "fr",  
          ?age >= 18,  
          ?age >= 21)
```

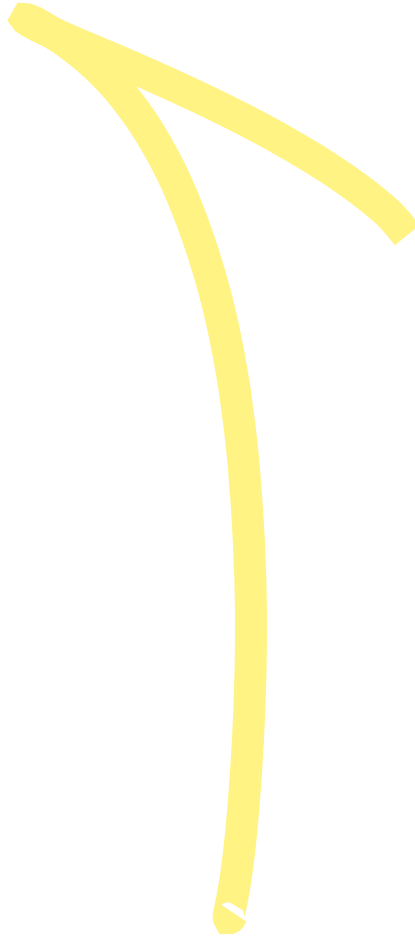
Coalesce

- Result of first expression that does not return an error (e.g. unbound variable)

filter **coalesce** (exp₁, .. exp_n)

Function

- isBlank(?x)
- isURI(?x)
- isLiteral(?x)
- bound(?x)



Function

- `datatype(?x)`
- `lang(?l)`
- `langMatches(lang(?l), "fr")`
- `str(<http://example.org>)`
- `uri("http://example.org")`
- `xsd:integer("123")`
- `xsd:string(12)`
- `strdt(str, datatype)`
- `strlang(str, lang)`

Function

- `now()`
- `year(?date)`
- `month(?date)`
- `day(?date)`
- `hours(?date)`
- `minutes(?date)`
- `seconds(?date)`
- ...

Function

- `contains(str1, str2)`
- `strstarts(str1, str2)`
- `strends(str1, str2)`
- `concat(str1, str2)`
- `substr(str, n)`
- `strlen(str)`
- `regex(str, ".*cnrs")`
- ...

Function

- `abs(?x)`
- `ceil(?x)`
- `floor(?x)`
- `round(?x)`
- `rand()`
- ...

Extension Function

- Application specific external function

```
prefix fun: <function://fr.geo.Extend>
select *
where {
    ?x geo:loc (?lon, ?lat)
    filter fun:locate(?lon, ?lat)
}
```

Example

- Teenager : age between 13 and 19

Example

- Teenager : age between 13 and 19

`?x foaf:age ?age`

`filter (?age >= 13 && ?age <= 19)`

Example

- Resource whose name length is less than 20

Example

- Resource whose name length is less than 20

```
?x foaf:name ?name  
filter (strlen(?name) < 20)
```

Example

- URI from inria

Example


- URI from inria

?x ?p ?y

filter regex(str(?x), "inria")

Query Form

Query Form

- 
- Two thick yellow curved lines, resembling stylized parentheses or a large right curly brace, are positioned to the right of the list items. They start near the top of the list and extend downwards, framing the text.
1. Select
 2. Ask
 3. Construct
 4. Describe

Select

- Return variable bindings

```
select * where {  
    ?x foaf:knows ?y  
}
```

Ask

- Return true/false

```
ask { ?x rdf:type ex:Yeti }
```

Construct

- Return new RDF graph
1. Instantiate construct pattern with every results
 2. Merge result triples into one new RDF graph

```
construct { ?x rdfs:seeAlso ?z }  
where { ?x foaf:knows ?y . ?y foaf:knows ?z }
```

Describe

- Return description of resource(s) as RDF graph

describe <http://example.org/John>



describe ?x

where { ?x foaf:name "John" }



Statement

Statement

1. Union
2. Optional
3. Minus
4. Filter (Not) Exists
5. Named Graph
6. Property Path
7. Nested Query
8. Bind
9. Values
10. Service

Union

- «Union» of results of alternative graph patterns
- One branch must match
- Both branches may match

```
{ ?x a ex:Good }
```

```
union
```

```
{ ?x a ex:Evil }
```


Statement

1. Union
- 2. Optional**
3. Minus
4. Filter (Not) Exists
5. Named Graph
6. Property Path
7. Nested Query
8. Bind
9. Values
10. Service

Optional

- Part of a query that is not mandatory
- Optional part may fail

`?x a foaf:Person`

`optional { ?x foaf:name ?name }`

1. `?x = us:John`
2. `?x = us:Jack ; ?name = "Jack"`

Optional

- Part of a query that is not mandatory
- Optional part may fail

`?x a foaf:Person`

`optional { ?x foaf:name ?name }`

`optional { ?x foaf:age ?age . filter (?age > 50) }`

Optional

- Part of a query that is not mandatory
- Optional part may fail

?x a foaf:Person

optional { ?x foaf:name ?name

optional { ?x foaf:age ?age . filter (?age > 50) }

}

Statement

1. Union
2. Optional
- 3. Minus**
4. Filter (Not) Exists
5. Named Graph
6. Property Path
7. Nested Query
8. Bind
9. Values
10. Service

Minus

- Negation
- Remove result of **minus pattern** from result of **pattern**

?x a foaf:Person

minus { ?x foaf:age ?age . filter (?age < 18) }

Minus

- There must be (at least) one variable in common

?x a foaf:Person

minus { **?x** foaf:age ?age . filter (?age < 18) }

Statement

1. Union
2. Optional
3. Minus
4. **Filter (Not) Exists**
5. Named Graph
6. Property Path
7. Nested Query
8. Bind
9. Values
10. Service

Filter Exists

- Check the **existence** of a pattern:
return true or false

?x a foaf:Person

filter exists { ?y foaf:knows ?x }



Filter Exists

- Check the **existence** of a pattern:
return true or false

```
?x a foaf:Person  
filter ( ?x = ex:JohnDoe ||  
  exists { ?y foaf:knows ?x } )
```

Filter Not Exists

- Check the **absence** of a pattern:
return true or false

?x a foaf:Person

filter **not exists** { ?y foaf:knows ?x }

Exercise

- Find persons *and* the resources they know
union the resources they are known by

Exercise

```
select * where {  
  ?x a foaf:Person  
  { ?x foaf:knows ?y } union { ?y foaf:knows ?x }  
}
```

Exercise

- Find persons except those who know John Doe

Exercise

```
select * where {  
  ?x a foaf:Person  
  minus { ?x foaf:knows us:JohnDoe }  
}
```

Exercise

Find persons and, if possible, persons they know and their name.

Exercise

```
select * where {  
  ?x a foaf:Person  
  optional {  
    ?x foaf:knows ?y  
    ?y foaf:name ?name  
  }  
}
```

Statement

1. Union
2. Optional
3. Minus
4. Filter (Not) Exists
- 5. Named Graph**
6. Property Path
7. Nested Query
8. Bind
9. Values
10. Service

RDF Dataset

- One default graph (mandatory)
- Several named graphs (optional)
- $\{ G, (uri_1, G_1), \dots (uri_n, G_n) \}$
- Basic Graph Pattern queries the default graph

RDF Dataset

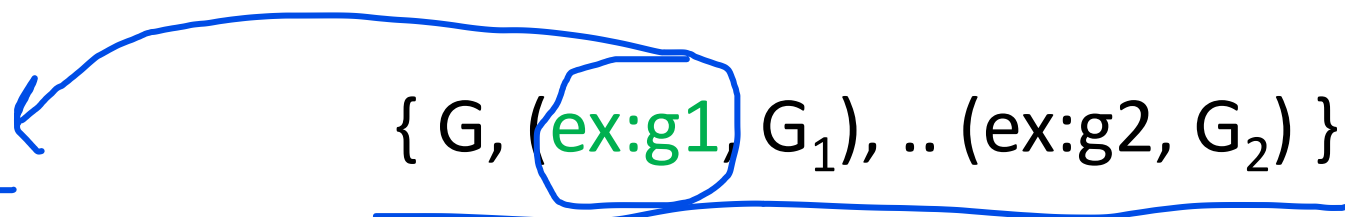
```
select *  
where {  
    ?x a foaf:Person  
}
```

```
{ G, (ex:g1, G1), ... (ex:g2, G2) }
```

From

- Redefine the default graph, for the query, either as:
 - one named graph
 - merge of several named graphs

```
select *  
from ex:g1 { G, (ex:g1, G1), .. (ex:g2, G2) }  
where {  
    ?x a foaf:Person  
}
```



From

- Redefine the default graph, for the query, either as:
 - one named graph
 - merge of several named graphs

select *

from ex:g1 { G, (ex:g1, G₁), .. (ex:g2, G₂) }

where {

?x a foaf:Person G₁

}

From

- Redefine the default graph, for the query, either as:
 - one named graph
 - merge of several named graphs

select *

from ex:g1

from ex:g2

where {

 ?x a foaf:Person

}

{ G, (ex:g1, G₁), .. (ex:g2, G₂) }

From

- Redefine the default graph, for the query, either as:
 - one named graph
 - merge of several named graphs

select *

from ex:g1

from ex:g2

where {

 ?x a foaf:Person

}

{ G, (ex:g1, G_1), .. (ex:g2, G_n) }



Named Graph Pattern

- Query one named graph

```
select * where {  
  graph ex:cnrs {  
    ?x foaf:knows ?y  
  }  
}
```

Named Graph Pattern

- Query all named graphs

```
select * where {  
  graph ?g {  
    ?x foaf:knows ?y  
  }  
}
```

From named

- Query specific named graphs, « *one by one* »

```
select *  
from named ex:cnrs  
from named ex:uca  
where {  
  
    graph ?g { ?x foaf:knows ?y }  
  
}
```

From and From named

```
select *  
from ex:inria  
from named ex:cnrs  
from named ex:uca  
where {  
    ?x a foaf:Person  
    graph ?g { ?x foaf:knows ?y }  
}
```

Statement

1. Union
2. Optional
3. Minus
4. Filter (Not) Exists
5. Named Graph
- 6. Property Path**
7. Nested Query
8. Bind
9. Values
10. Service

Property Path

Find resources related by several triples:

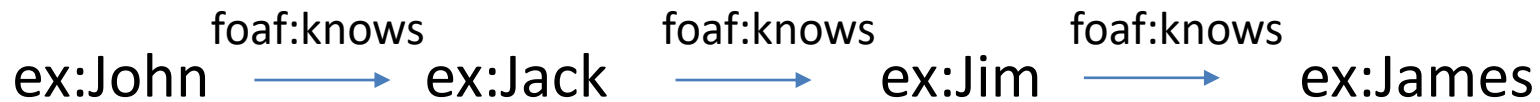
ex:John $\xrightarrow{\text{foaf:knows}}$ ex:Jack $\xrightarrow{\text{foaf:knows}}$ ex:Jim $\xrightarrow{\text{foaf:knows}}$ ex:James

ex:John **foaf:knows** ?y

(1) ?y = ex:Jack

Property Path

Find resources related by several triples:



ex:John **foaf:knows/foaf:knows** ?y

(1) ?y = ex:Jim

Property Path

Find resources related by several triples:

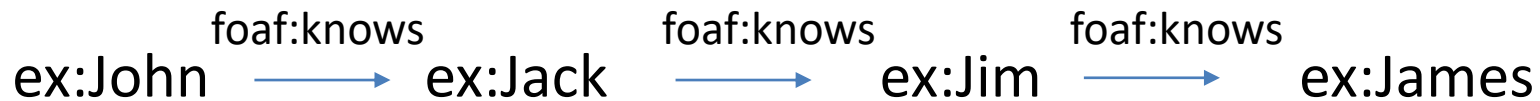
ex:John $\xrightarrow{\text{foaf:knows}}$ ex:Jack $\xrightarrow{\text{foaf:knows}}$ ex:Jim $\xrightarrow{\text{foaf:knows}}$ ex:James

ex:John $\text{foaf:knows/foaf:knows/foaf:knows}$?y

(1) ?y = ex:James

Property Path

Find resources related by several triples:



ex:John **foaf:knows+** ?y

- (1) ?y = ex:Jack
- (2) ?y = ex:Jim
- (3) ?y = ex:James

Property Path Language

?x exp ?y

exp ::=

uri : property

!uri : negation

exp/exp : sequence

exp|exp : alternative

exp? : optional

exp* : zero or several

exp+ : one or several

^exp : reverse

Property Path: Reverse

$?x \text{ } ^{(p/q)} \text{ } ?y$

$::=$

$?y \text{ } p/q \text{ } ?x$

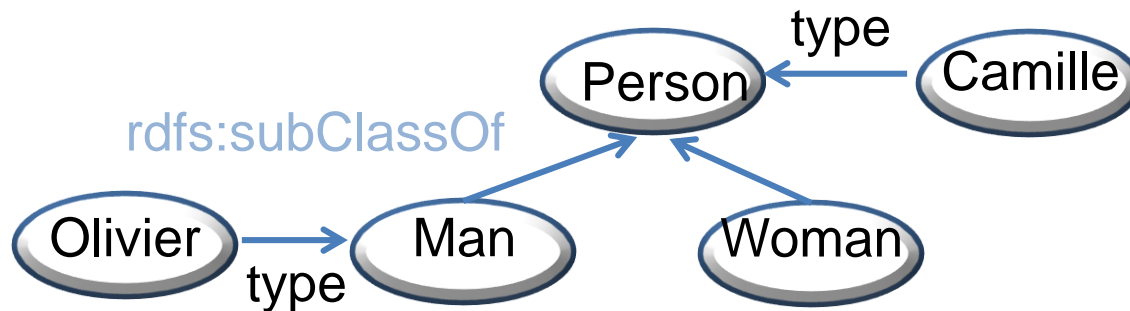
$?x \text{ } ^{\text{exp}} \text{ } ?y$

$::=$

$?y \text{ } \text{exp} \text{ } ?x$

Path Example

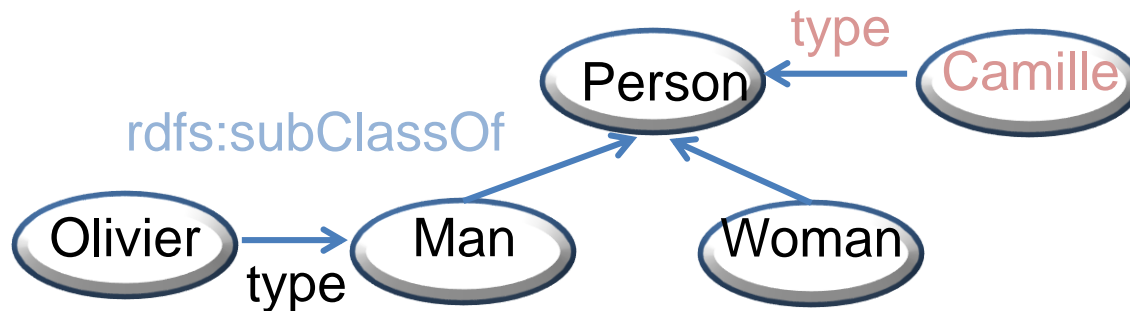
- Emulate class subsumption



Path Example

- Emulate class subsumption

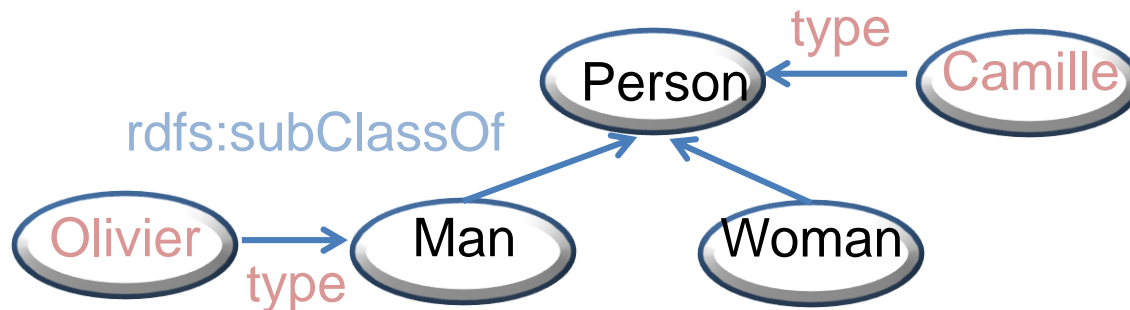
?x rdf:type foaf:Person



Path Example

- Emulate class subsumption

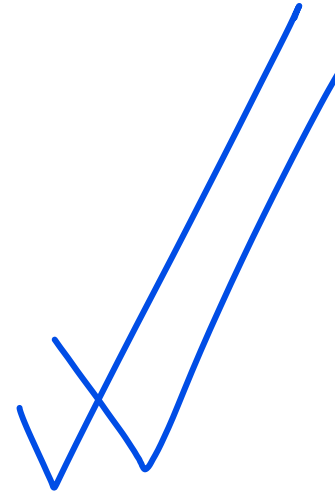
`?x rdf:type/rdfs:subClassOf* foaf:Person`



Path Example

- Enumerate RDF list elements

`?list rdf:rest*/rdf:first ?e`



Path Example

- Resources related by several foaf:knows and/or rdfs:seeAlso

Path Example


- Resources related by several foaf:knows and/or rdfs:seeAlso

`?x (foaf:knows|rdfs:seeAlso)+ ?y`


Path Example

- Enumerate list elements with position

```
select ?val (count(?mid) as ?pos) where {  
  ?list  rdf:rest* ?mid  
  ?mid  rdf:rest* ?node  
  ?node rdf:first  ?val  
}  
group by ?node ?val
```



Statement

1. Union
2. Optional
3. Minus
4. Filter (Not) Exists
5. Named Graph
6. Property Path
- 7. Nested Query** 
8. Bind
9. Values
10. Service

Nested Query

- Subquery within a query

```
select * where {  
  {select ?x (?w * ?l as ?area)  
   where { ?x ex:width ?w ; ex:length ?l }}  
  ?z ex:area ?area  
}
```

Statement

1. Union
2. Optional
3. Minus
4. Filter (Not) Exists
5. Named Graph
6. Property Path
7. Nested Query
- 8. Bind**
9. Values
10. Service

Bind

- Bind result of expression to variable

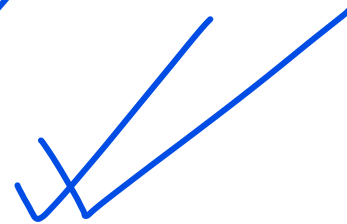
`bind (exp as var)`



Bind

?x geo:width ?w ; geo:length ?l

bind (?w * ?l **as** ?area)



Statement

1. Union
2. Optional
3. Minus
4. Filter (Not) Exists
5. Named Graph
6. Property Path
7. Nested Query
8. Bind
- 9. Values**
10. Service

Values

- Focus variable(s) on predefined constant values

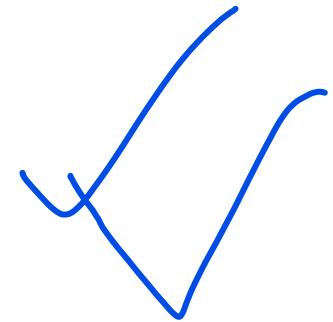
```
select * where {  
    ?x rdfs:label ?name  
}
```

```
values ?name { "Blue" "Red" "Yellow" }
```

Values

- Focus variable(s) on predefined constant values

```
select * where {  
    values ?name { "Blue" "Red" "Yellow" }  
    ?x rdfs:label ?name  
}
```



Statement

1. Union
2. Optional
3. Minus
4. Filter (Not) Exists
5. Named Graph
6. Property Path
7. Nested Query
8. Bind
9. Values
- 10. Service**

Service

SPARQL 1.1. Federated Query



Service

- Query remote SPARQL endpoint

```
select * where {  
  service <http://fr.dbpedia.org/sparql> {  
    ?x rdfs:label "Nice"@fr ;  
    rdf:type ?t  
  }  
}
```

Service

- Query remote SPARQL endpoint
- Join service results with local results

```
select * where {  
  service <http://fr.dbpedia.org/sparql> {  
    ?x rdfs:label "Nice"@fr ;  
    rdf:type ?t  
  }  
  ?y a ?t  
}
```

Service


- Query remote SPARQL endpoint with nested query

```
select * where {  
  service <http://fr.dbpedia.org/sparql> {  
    select * where {  
      ?x rdfs:label "Nice"@fr ;  
      rdf:type ?t  
    }  
    limit 50  
  }  
}
```

back to

SPARQL 1.1. Query Language

Solution Sequence Modifiers

1. Distinct
 2. Order By
 3. Limit
 4. Offset
- 

Distinct

- Remove similar results with same values for same variables

```
select ?x ?z
where {
    ?x foaf:knows ?y .
    ?y foaf:knows ?z
}
```

1. ?x = ex:John ; ?z = ex:Jack
2. ...
3. ?x = ex:John ; ?z = ex:Jack

Distinct

- Remove similar results with same values for same variables

```
select distinct ?x ?z
where {
    ?x foaf:knows ?y .
    ?y foaf:knows ?z
}
```

1. ?x = ex:John ; ?z = ex:Jack
2. ...
- ~~3. ?x = ex:John ; ?z = ex:Jack~~

Order By

- Sort results by numbers, strings, dates, ...

```
select * where {  
  ?x foaf:name ?name ;  
    foaf:age ?age  
}  
order by desc(?age) ?name
```

Limit, Offset

- Limit number of results
- Slice results with an offset

```
select * where {
```

```
    ...
```

```
}
```

```
limit 10
```

```
offset 10
```

Select Expression and Aggregates

Select Expression

- Complete the select clause with expressions

```
select ?s (?w * ?l as ?area)  
where {  
    ?s ex:width ?w ; ex:length ?l  
}
```

Aggregates

- Combine several results into an aggregate value.

```
select (avg(?p) as ?avg) where {  
    ?x ex:price ?p  
}
```

1. count
2. sum
3. avg
4. min
5. max
6. group_concat
7. sample

Aggregates and Group By


- Group results with similar values, aggregate in each group

```
select ?x (avg(?s) as ?score)
where {
    ?x ex:score ?s
}
group by ?x
```

Aggregates, Group By and Having

- Group results with similar values, aggregate in each group, filter aggregate values

```
select ?x (avg(?s) as ?score)
where {
    ?x ex:score ?s
}
group by ?x
having (?score >= 10)
```



DESIGN PATTERNS

Negation as Failure

```
?x a foaf:Person  
optional { ?x dc:creator ?doc }  
filter (! bound(?doc))
```

Negation : take care

- Find people that are not fisherman

ex:John a foaf:Person , ex:Fisherman .

```
select * where {  
    ?x a ?t  
    filter (?t != ex:Fisherman)  
}
```

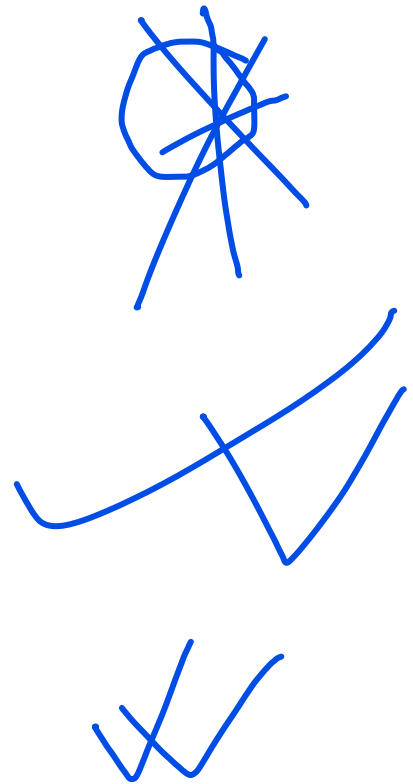
☹ ?x = ex:John ; ?t = foaf:Person

Negation : take care

- Find people that are not fisherman

ex:John a foaf:Person , ex:Fisherman .

```
select * where {  
  ?x a ?t  
  minus { ?x a ex:Fisherman }  
}
```



Named Graph as Context

```
select *  
where {  
  
    graph ?g {  
        ?x ex:workAt <http://www.cnrs.fr>  
    }  
}
```

Named Graph as Context

```
select *  
where {  
    ?g a ex:Context  
    graph ?g {  
        ?x ex:workAt <http://www.cnrs.fr>  
    }  
}
```

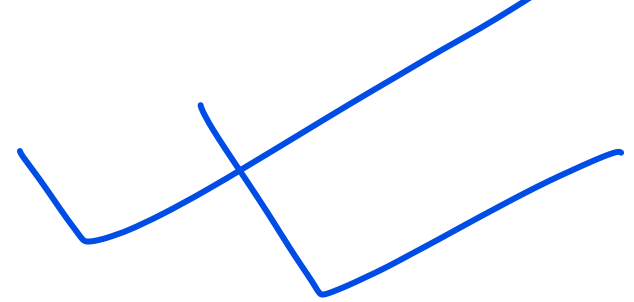

Named Graph as Context

```
select *  
where {  
    ?g a ex:Context ; ex:date 2001  
    graph ?g {  
        ?x ex:workAt <http://www.cnrs.fr>  
    }  
}
```

Named Graph as Context

```
select *  
where {  
    ?g1 a ex:Context ; ex:date 2001 .  
    ?g1 rdfs:seeAlso ?g2  
    graph ?g2 {  
        ?x ex:workAt <http://www.cnrs.fr>  
    }  
}
```

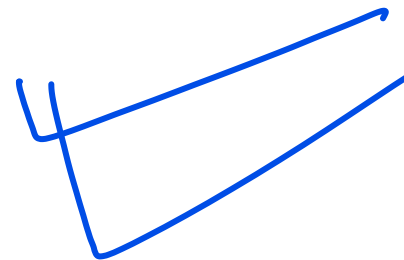
Distinct distinct



```
select distinct ?x ?y
where {
    ?x foaf:knows/foaf:knows ?y
}
```

```
?x = ex:John ; ?y = ex:Patty
?x = ex:Patty ; ?y = ex:John
```

Distinct distinct



```
select distinct ?x ?y
where {
    ?x foaf:knows/foaf:knows ?y
    filter (?x < ?y)
}
```

?x = ex:John ; ?y = ex:Patty

~~?x = ex:Patty ; ?y = ex:John~~

SPARQL 1.1 QUERY RESULT

SPARQL Query Result Formats

- XML
- JSON
- CSV, TSV

SPARQL 1.1 UPDATE

SPARQL Update

Manage (modify) the content of RDF Datasets

- Load
- Delete/Insert Data
- Delete/Insert Where
- Copy, Move, Add

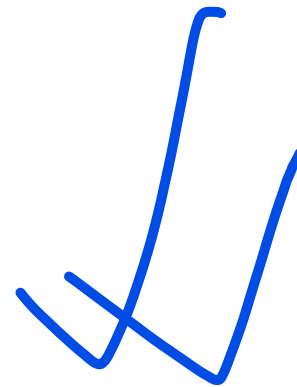
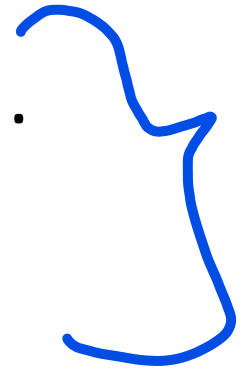
Load RDF document

load <http://example.org/data.ttl>

load <http://example.org/data.ttl> into graph ex:g

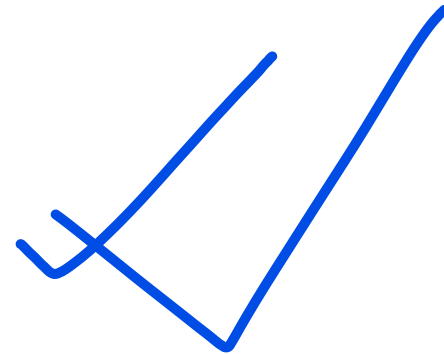
Insert Data

```
insert data {  
    ex:John a foaf:Person ; foaf:age 18 .  
    ex:Jill a foaf:Person ; foaf:age 81 .  
}
```



Delete Data

```
delete data {  
    ex:John a foaf:Person ; foaf:age 18 .  
    ex:Jill a foaf:Person ; foaf:age 81 .  
}
```



Delete Data

```
delete data {  
    [ ] a foaf:Person ; foaf:age 18.  
    ex:Jill a foaf:Person ; foaf:age 81 .  
}
```

No Blank Node in a delete clause !!!



Delete Where

```
delete {  
    ?x foaf:age ?age  
}  
where {  
    ?x a foaf:Person ; foaf:age ?age  
    filter (?age < 0)  
}
```



Delete Where

```
delete {  
    [ ] foaf:age ?age  
}  
where {  
    ?x a foaf:Person ; foaf:age ?age  
    filter (?age < 0)  
}
```

No Blank Node in a delete clause !!!



Delete Where

```
delete {  
    ?x foaf:age ?age  
}  
where {  
    ?x a foaf:Person ; foaf:age ?age  
    filter (?age < 0)  
}  
  
(1) ?x = ex:John  
(2) ?x = _:b1
```

Insert Where

```
insert {  
    ?x foaf:mail ?mail  
}  
where {  
    ?x ex:firstName ?f ; ex:lastName ?l  
    bind (concat(?f, ".", ?l, "@acme.com")  
as ?mail)  
}
```


Insert Where

```
insert {  
  [ ] foaf:mail ?mail ✓  
}  
where {  
  ?x ex:firstName ?f ; ex:lastName ?l  
  bind (concat(?f, ".", ?l, "@acme.com")  
    as ?mail)  
}
```

Update: Delete Insert Where

```
delete { ?x foaf:age ?age }  
insert { ?x foaf:age ?int }  
where {  
    ?x foaf:age ?age  
    filter (datatype(?age) = xsd:string)  
    bind (xsd:integer(?age) as ?int)  
}
```

1. Compute solutions $\{S_1, \dots S_n\}$ of the where clause
2. Apply delete on $\{S_1, \dots S_n\}$
3. Apply insert on $\{S_1, \dots S_n\}$

Named Graph

```
insert {  
    graph ex:info { ?x foaf:mail ?mail }  
}  
where {  
    ?x ex:firstName ?f ; ex:lastName ?l  
    bind (concat(?f, ".", ?l, "@acme.com")  
    as ?mail)  
}
```

Others

- Copy
- Move
- Add

SPARQL 1.1. PROTOCOL

SPARQL Protocol

- SPARQL Endpoint
 - Triple Store embedded in an HTTP server
- SPARQL Protocol
 - Interact with SPARQL endpoint by means of HTTP

SPARQL Protocol

- Interact with SPARQL endpoint by means of HTTP

`http://dbpedia.org/sparql?query=
select * where { ?x rdfs:label "Paris"@fr }`

`http://example.org/sparql?update=
load <http://example.org/data.ttl>`