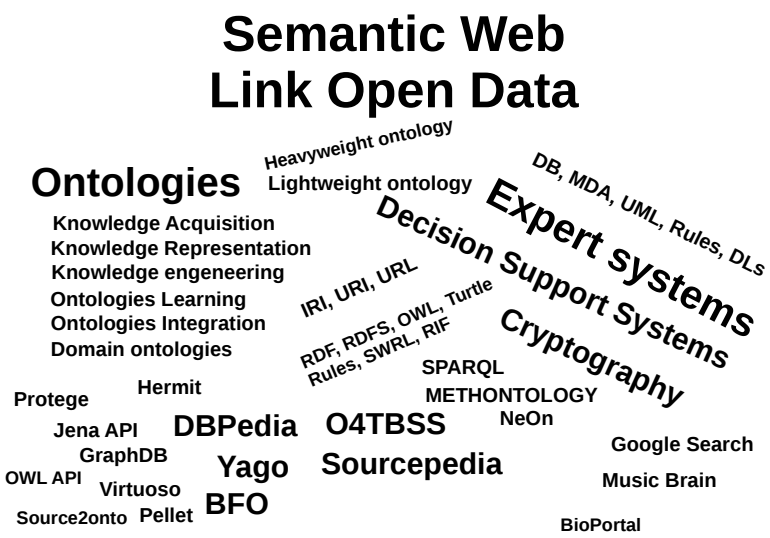
# Semantic Web and knowledge engineering

## Queries Languages for the Semantic Web




Dr. Azanzi Jiomekong

**University of Yaounde I**
**Computer Science Department**

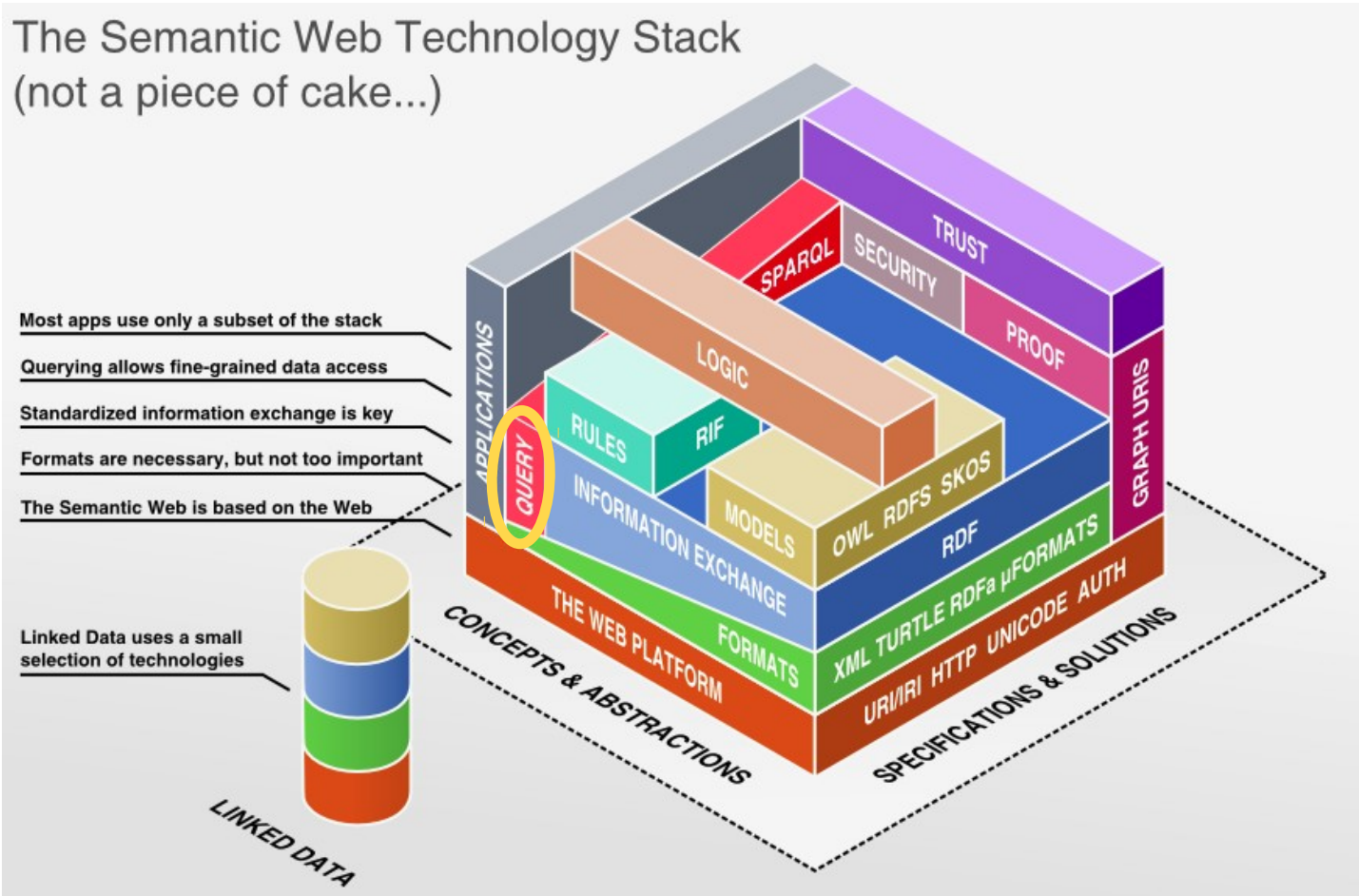
2 mars 2022

# Questions


- Consider we have a large RDF(S) knowledge base or a large knowledge graph :
  - How do we access the knowledge from RDF/RDFS Knowledge Bases ? Manual Parsing/Reading and Combination of RDF triples is complex
  - How do we query this knowledge base ?
  - How do we extract knowledge ?
  - How do we ask question to this knowledge base ?

**The responses to these questions are given in this course**

# Queries languages for the SW



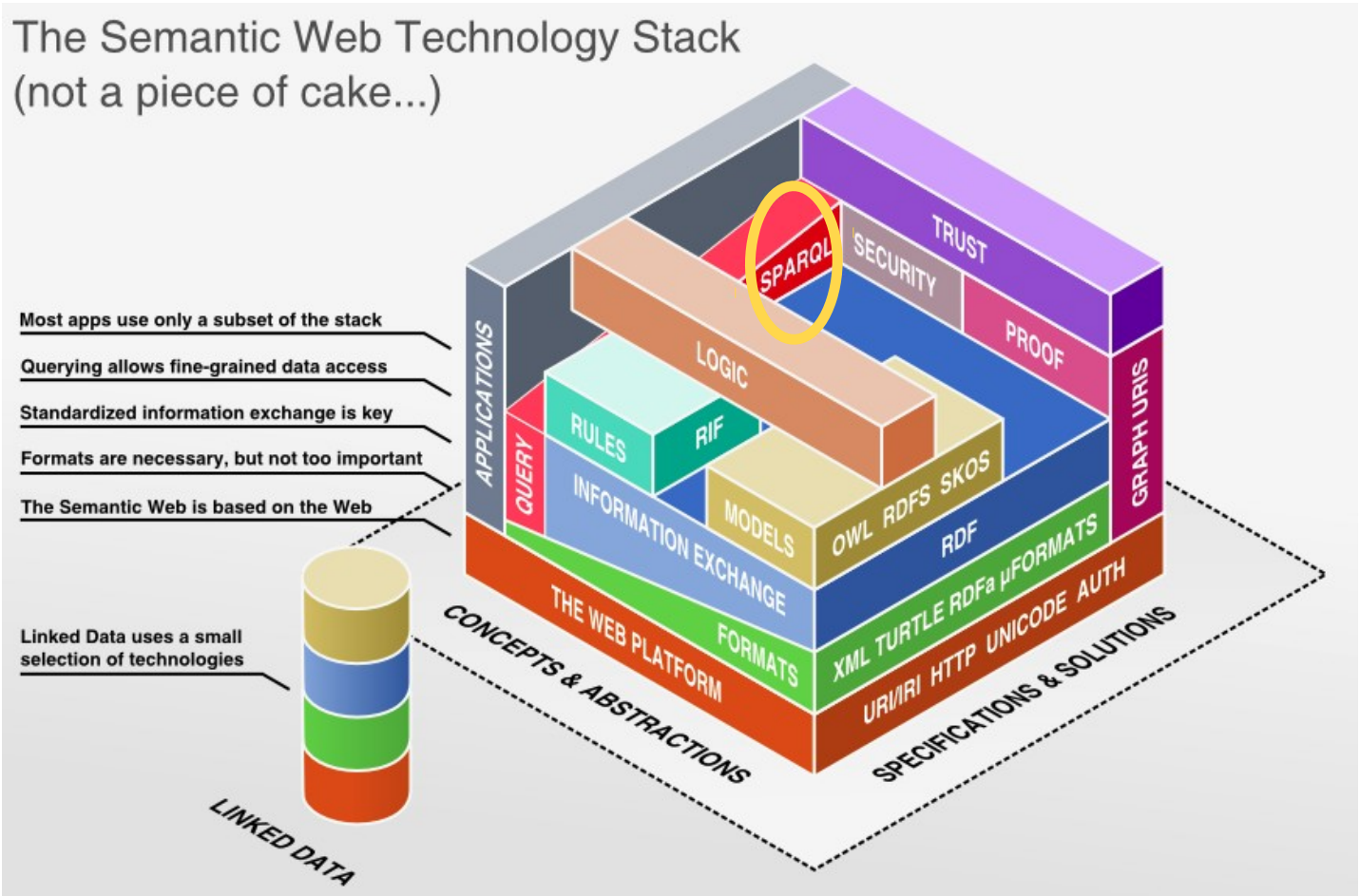
# Queries languages for the SW


- DL Query Language : used in Protege for querying and searching
  `https://ontology101tutorial.readthedocs.io/en/latest/DL_QueryTab.html`

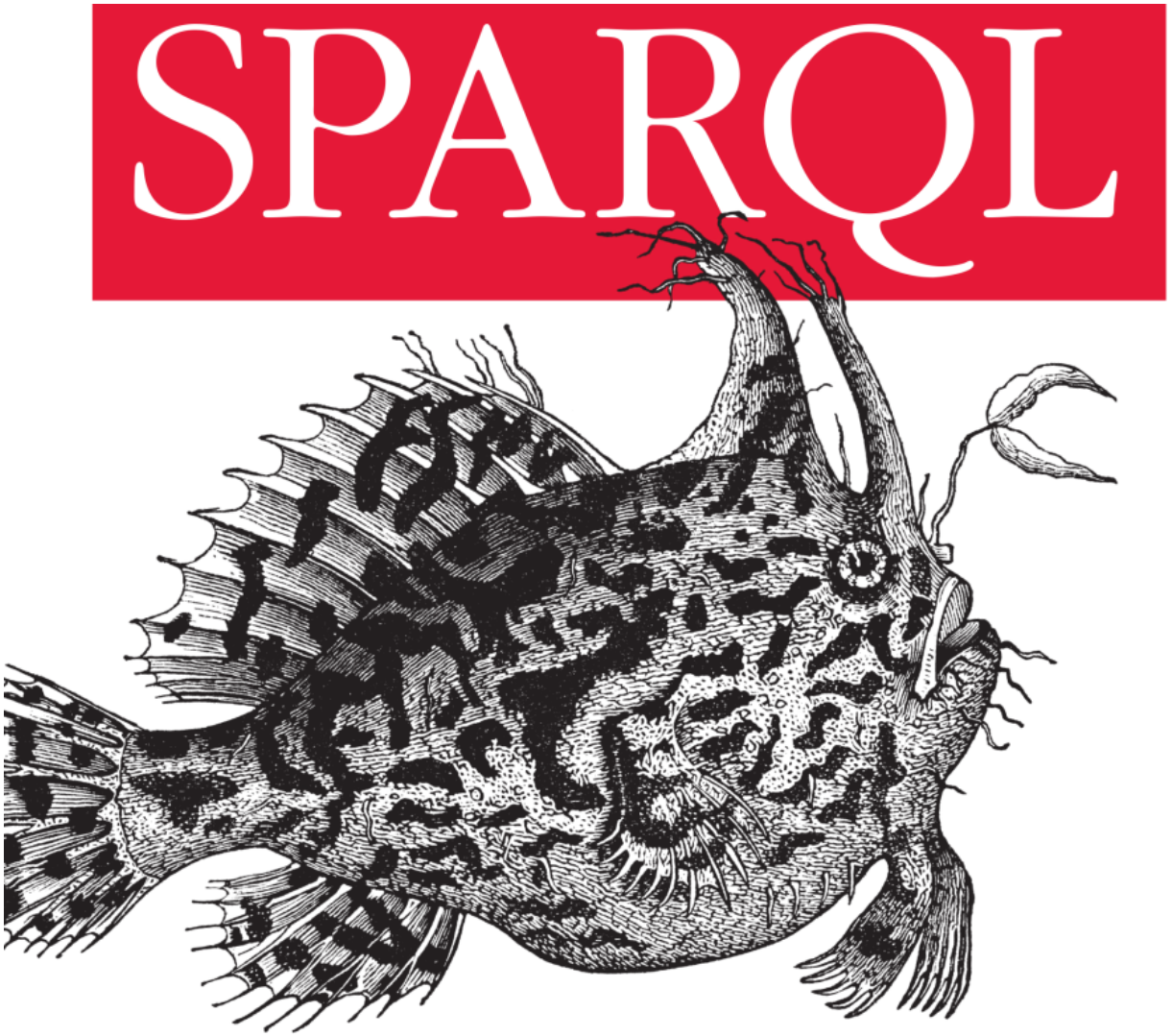
- SPARQL :
  - SPARQL 1.0 : W3C Recommendation - 15 January 2008
    `https://www.w3.org/TR/rdf-sparql-query/`
  - SPARQL 1.1 : W3C Recommendation - 21 March 2013
    `https://www.w3.org/TR/sparql11-query/`

- SQWRL :
  `https://github.com/protegeproject/swrlapi/wiki/SQWRL`
  - Semantic Query-Enhanced Web Rule Language
  - SWRL-based query language
  - Provides SQL-like operators for extracting information from OWL ontologies

# SPARQL


The Semantic Web Technology Stack
(not a piece of cake...)
Most apps use only a subset of the stack
Querying allows fine-grained data access
Standardized information exchange is key
Formats are necessary, but not too important
The Semantic Web is based on the Web
Linked Data uses a small selection of technologies
LINKED DATA
APPLICATIONS
QUERY
SPARQL
TRUST
SECURITY
PROOF
LOGIC
RULES
RIF
INFORMATION EXCHANGE
MODELS
OWL RDFS SKOS
RDF
GRAPH URIS
THE WEB PLATFORM
FORMATS
XML TURTLE RDFa μFORMATS
URI/IRI HTTP UNICODE AUTH
CONCEPTS & ABSTRACTIONS
SPECIFICATIONS & SOLUTIONS

# SPARQL



# SPARQL


- **SPARQL : P**rotocol **a**nd **R**DF **Q**uery **L**anguage :
  - A **Query Language** for RDF Graph Transversal
    *SPARQL Query Language Specification*
  - A **Protocol Layer**, to use SPARQL via HTTP
    *SPARQL Protocol for RDF Specification*
  - An **XML Output Format Specification** for SPARQL Queries
    *SPARQL Query XML Results Format*
  - W3C Standard
  - Inspired by SQL

# SPARQL


- Query language for the Semantic Web
- Covers starting from RDF over RDFS (see Semantic Web Technology stack)
- Used to ask questions from an RDF knowledge base and generate the answer
- Full data manipulation language (data access and manipulation)
- Globally, SPARQL :
  - Query language for RDF graphs
  - Protocol on top of HTTP specifying how to access and query a SPARQL end point
  - Specify a XML output format specification for SPARQL queries

# SPARQL

## Usage


- **Extraction** of data as :
  - URIs, Blank Nodes, types and untyped Literals
  - RDF Subgraphs
- **Exploration** of data via Query for unknown relations
- Execution of **complex Join Operations** on heterogeneous databases in a single query
- **Transformation** of RDF Data from one vocabulary into another
- **Construction** of new RDF Graphs based on RDF Query Graphs

# SPARQL

## Usage


- Aggregate functions, subqueries, negations, project expressions, property paths
- Logical **Entailment** for RDF, RDFS, OWL, Direct and RDF-Based Semantics entailment and RIF Core entailment
- **Update of RDF Graphs** as a full data manipulation language
- **Discovery of information** about the SPARQL service
- **Federated Queries** distributed over different SPARQL

# SPARQL


To query a RDF/RDFS knowledge base :

- Define SPARQL variables
- Variables cover facts, things, resources we are getting out of the knowledge base
- Each variable is preceded by a question mark

# SPARQL

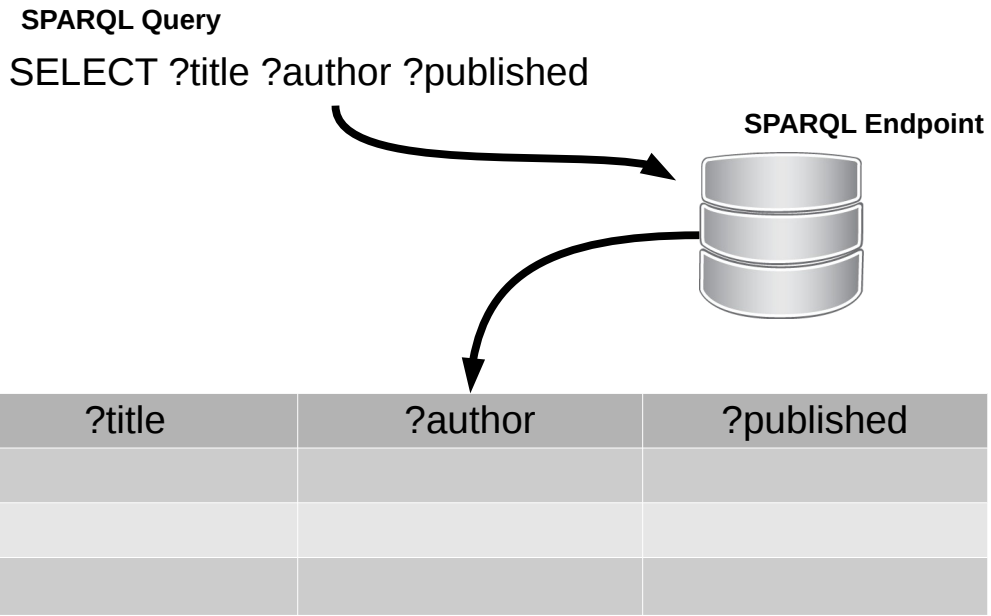

- SPARQL **Variables** are bound to RDF terms
  e.g. ?journal, ?disease, ?price

- A **Query** for variables is performed via SELECT statement
  e.g. SELECT ?title ?author ?published

- A SELECT statement returns **Query Results** as a table

| ?title | ?author | ?published |
|---|---|---|
| The Devil and Miss Prym | Paulo Coelho | 2000 |
| Unbroken : A World War II Story of Survival, Resilience, and Redemption | Laura Hillenbrand | 2010 |
| Rich Dad Poor Dad : What the Rich Teach Their Kids about Money That the Poor and Middle Class Do Not ! | Robert Kiyosaki | 1997 |

# SPARQL




Endpoints :

- The place the API send the request and where the resource lives
- Involve the customization of HTTP requests to call a specific resource or retrieve specific data
- Two users may have different endpoint to access to the same resource


# SPARQL

## Graph Pattern

- SPARQL is based on **RDF Turtle serialization** and **basic graph pattern matching**

- A Graph Pattern (Triple Pattern) is a RDF Triple that contains variables at any arbitrary place (Subject, Predicate, Object)

- Each triple is closed by a period

- **(Graph) Triple Pattern = Turtle + Variables**
  **Example :**
  *Look for countries and their capitals*
  *?country geo :capital ?capital .*


# SPARQL

## Graph Pattern

Triple Pattern

?country **geo:capital** ?capital .

RDF Graph

dbpedia:Cameroon rdf:type dbpedia-owl:Country .

dbpedia:Cameroon **geo:capital** "Yaounde" .

dbpedia:Cameroon dbprop:language "Français" .

dbpedia:Cameroon dbprop:language "English" .

dbpedia:Senegal rdf:type dbpedia-owl:Country .

dbpedia:Senegal **geo:capital** "Dakar" .

dbpedia:Senegal dbprop:language "Français" .

...

# SPARQL

## Graph Pattern


- **More Examples :**
  **Given a FOAF URI, find the name of a person :**
  $< http : //facscience - UY1.uninet.cm/id/azanzijiomekong >$
  $foaf : name?surname.$

- **Which persons have the family name "Atemengue ?"**
  ?person pers :familyName "Atemengue" .

**Note :** FOAF is used to describe knowledge about persons and social network information of the person


# SPARQL

## Complex Query Pattern

- Graph Pattern can be combined to form **complex (conjunctive) queries** for RDF graph transversal

- *Find countries, their capitals and their population* :
  ?country geo :capital ?capital .
  ?country geo :population ?population .

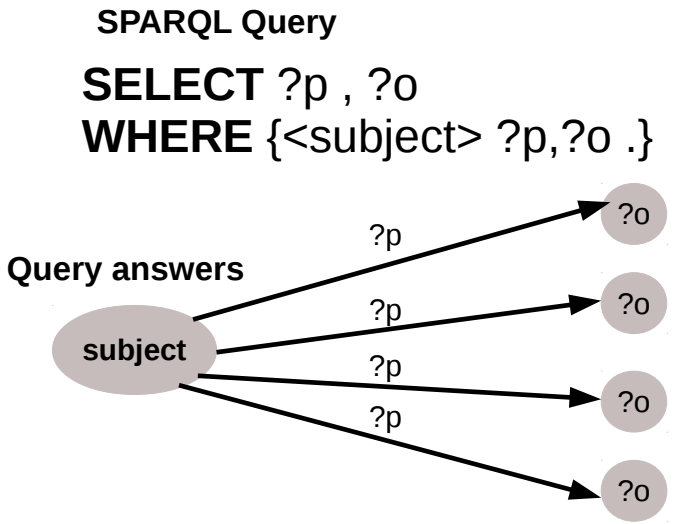
- Given a FOAF URI, find the name of a person and her friends :

```
<http://facscience-UY1.uninet.cm/id/azanzijiomekong> foaf:name ?surname ;
                                                     foaf:knows ?friend .
        ?friend foaf:name ?friend_surname .
```

# SPARQL

## SPARQL Query Format

- Triple in **SELECT** part defines the variables that fit the patterns in the query
- Triples in **WHERE** part define graph query with variables ?p and ?o
- Query returns table with matching ?p, ?o pairs




# SPARQL

## SPARQL Query Format

- **WHERE :** specifies graph pattern to be matched
- **PREFIX :**
  - specifies one or more namespaces for using compact URIs (CURIEs)
  - did not end with a period
  - prefixes can be found at : `prefix.cc`
- **FROM :** specifies one or more RDF source graphs
- **BASE :** defines a base URI


# SPARQL

## Popular prefixes

- dc (Dublin Core) : identifying or characterizing bibliography
- foaf (Friend Of A Friend) :
- SIOC () :
- DBpedia :
- DBLP :
- etc.


# SPARQL

## SPARQL Query Format

```
PREFIX fsuy1: <http://facscience-uy1.uninet.cm/swcapplication#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?lecture ?manager
FROM <...>
WHERE {
        ?x rdf:type fsuy1:Lecture .
        ?x rdfs:label ?lecture .
        ?y rdf:type fsuy1:Staff .
        ?y rdfs:label ?manager .
        ?x fsuy1:isManagedBy ?y .
}
```

- *Search all lectures and their managers :*
  - Select for the variable 'x' something that has the type "Lecture"
  - Then, select the label of this lecture
  - Then, select somebody from the staff
  - Then, select the label of the staff member
  - And in the end, we want to combine the lectures and the staff members
  - Take into combination the lectures and there according to the managers

# SPARQL


```
PREFIX fsuy1: <http://facscience-uy1.uninet.cm/swcapplication#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?lecture ?manager
FROM <...>
WHERE {
        ?x rdf:type fsuy1:Lecture ;
                ?x rdfs:label ?lecture .
        ?y rdf:type fsuy1:Staff ;
                ?y rdfs:label ?manager .
        ?x fsuy1:isManagedBy ?y .
}
ORDER By DESC {?manager}
LIMIT 10
OFFSET 10
```

- *Search all lectures and their managers* ordered by *managers in descending order and* limit *the results to the first 10* starting *the list at position 10*
  - The result should be ordered in a descending order according to the name of the manager
  - Limit the output to the first 10 results
  - Start by the 10th result


# SPARQL

## Blank Nodes in SPARQL Queries

- As Subject or object of a triple pattern
- "Non selectable" variables

```
PREFIX fsuy1: <http://facscience-uy1.uninet.cm/swcapplication#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name
FROM <...>
WHERE {
        _:x rdf:type fsuy1:Lecture ;
                fsuy1:isManagedBy [rdfs:label ?name] .
}
```

- *Select the names of the managers of lectures*
  - They are identified by a Blank node
  - Look for Blank nodes of type lecture
  - The seeking Blank nodes should have the property "isManagedBy"

Blank node identifier might also occurs in the results


# SPARQL

## SPARQL Query Format : FILTER

- FILTER is used to :
  - Put some conditions to the graph pattern
  - Specify constraints on the results
- Consider the example with :
  - Three prefixes
  - Books with the title from the dc name space and with a specific price.
- Question : what are the books that cost less than 30.5 ?


# SPARQL

## SPARQL Query Format : FILTER

- The keyword **FILTER** specifies constraints for the result
  - FILTER expressions contain operators and functions
  - FILTER can NOT assign/create new values

```
#Default Graph (stored at http://example.org/book)
 @prefix dc: <http://purl.org/dc/element/1.1/> .
 @prefix : <http://example.org/book> .
 @prefix ns: <http://example.org/ns#> .

 :book1 dc:title "SPARQL Tutorial" .
 :book1 ns:price 42 .
 :book2 dc:title "Semantic-Aware Software" .
 :book2 ns:price 125 .
```

```
PREFIX dc: <http://purl.org/dc/element/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
FROM <http://example.org/book>
WHERE {
        ?x ns:price ?price .
                FILTER (?price < 35)
                ?x dc:title ?title .
 }
```


# SPARQL

## SPARQL Operators

Unary Operators in constraints

| **Operator** | **Type(A)** | **Result Type** |
|---|---|---|
| !A | xsd :boolean | xsd :boolean |
| +A | numeric | numeric |
| -a | numeric | numeric |
| BOUND(A) | variable | xsd :boolean |
| isURI | RDF term | xsd :boolean |
| isBLANK(A) | RDF term | xsd :boolean |
| isLITERAL(A) | RDF term | xsd :boolean |
| STR(A) | literal/URI | simple literal |
| LANG(A) | literal | simple literal |
| DATATYPE(A) | literal | URI |


# SPARQL

## SPARQL Operators

- Logical connectives && and || for *xsd* : *boolean*
- Comparison operators $=, !=, <, >, <=, >=$ for numeric, datatypes, *xsd* : *dateTime*, *xsd* : *string*, *xsd* : *boolean*
- Comparison operators $=$ and $!=$ for other datatypes
- Arithmetic operators $+, -, *, /$ for numeric datatypes

An in addition :

- **REGEX(String, Pattern)** or **REGEX**(String, Pattern, Flags)
- **sameTERM**(A,B) : to compare two terms
- **langMATCHES**(A,B) : compare what the two languages of the two literals really match


# SPARQL

## Evaluation of FILTER constraints

- FILTER constraints :
  - Based on a ternary logic = three value logic
  - Each part connected to an "or", an "and" can be either true, false, or there migh be an error
  - evaluated in 3-values logic : true, false, and error

| A | B | A\|\|B | A&&B |
|---|---|---|---|
| T | T | T | T |
| T | F | T | F |
| F | T | T | F |
| F | F | F | F |
| T | E | T | E |
| E | T | T | E |
| F | E | E | F |
| E | F | E | F |
| E | E | E | E |

| A | !A |
|---|---|
| T | F |
| F | T |
| E | E |


# SPARQL

## Query Format : OPTIONAL

Let's consider a knowledge base with two prefixes (the foaf prefix and the rdf prefix) and some facts according to two persons in this knowledge base.

- Question : What are the names and the mailboxes of each persons ?
- Problem :
  - Some persons don't have the mailbox
    $\longrightarrow$ A simple request cannot help to get the mailboxes of the other
- Solution : use the "OPTIONAL" keyword to express some patterns that whenever they are available, will also be processed


# SPARQL

## Query Format : OPTIONAL

- The keyword **OPTIONAL** :
  - Selects optional elements from the RDF graph
  - Complies to a Left Outer Join
  - Used to put the optional restrictions on to the graph pattern.

```
#Default Graph (stored at http://example.org/addresses)
@prefix foaf: <http://xmlns.com/foaf/0.1> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

_:a rdf:type    foaf:Person .
_:a foaf:name   "Maxim" .
_:a foaf:mbox   <mailto:maxim@example.com> .
_:a foaf:mbox   <mailto:maxim@facscience-uy1.uninet.cm> .

_:b rdfs:type   foaf:Person .
_:b foaf:name   "Ronald"
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1>
SELECT ?name ?mbox
FROM <http://example.org/addresses>
WHERE {
        ?x foaf:name ?name .
                OPTIONAL {?x foaf:mbox ?mbox}
 }
```


# SPARQL

## Query Format : UNION

The keyword **UNION**
$\longrightarrow$ Allows for alternatives (logical disjunction) $\longrightarrow$ Correspond to the "or" expression

- Let's consider the knowledge base containing books from two Dublin Core prefixes.
- Question : What are all the titles of all the books ?
- Problem : Replying to the previous question is not possible with a single graph expression
- Solution : Combine the answer comming from "dc10" and "dc11" using an "or" condition


# SPARQL

## Query Format : UNION

```
#Default Graph (stored at http://example.org/book)
@prefix dc10: <http://purl.org/dc/element/1.0/> .
@prefix dc11: <http://purl.org/dc/element/1.1/> .

_:a dc10:title "RDF graph tutorial" .
_:a dc10:creator       "Teko" .

_:b dc11:title "Music building tutorial"
_:b dc11:creator       "Excel" .

_:c dc10:title "Integrating Apache Sorl in a Triple Store" .
_:c dc11:creator       "Donald, Regis" .
```

```
PREFIX dc10: <http://purl.org/dc/element/1.0/>
PREFIX dc11: <http://purl.org/dc/element/1.1/>
SELECT ?title
FROM <http://example.org/book>
WHERE {
        {?book dc10:title ?title}
        UNION
        {?book dc11:title ?title}
}
```

# SPARQL

## Query Format : Negation


- Complies to : NOT, EXIST in SQL
- Some condition should not be filled
- Question : Give me all persons, but only those that do not provided a date.
- Solution : Use the "bound" condition in which the variable "date" is not bound


# SPARQL

## Query Format : Negation

```
@prefix foaf: <http://xmlns.com/foaf/0.1> .
@prefix dc: <http://purl.org/dc/element/1.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

_:a foaf:givenName "Maxim" .

_:b foaf:givenName "Ronald"
_:b dc:date "2021-05-04T04:04:04z"^^xsd:dateTime .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1>
PREFIX dc: <http://purl.org/dc/element/1.1/>

SELECT ?name
WHERE {
        ?x foaf:givenName ?name .
                OPTIONAL {?x dc:date ?date} .
                FILTER (!bound(?date))
 }
```

# SPARQL

## Query Format


- Many graphs can be refer in the same SPARQL query
  $\longrightarrow$ The graphs are named graphs

- Two types of named graphs :
  1. Default graph
  2. Graphs that are explicitly address within a query

- "FROM" clause : use to select graph pattern with more than only one graph

- "Graph" keyword : used to specify all the graph we are refering to


# SPARQL

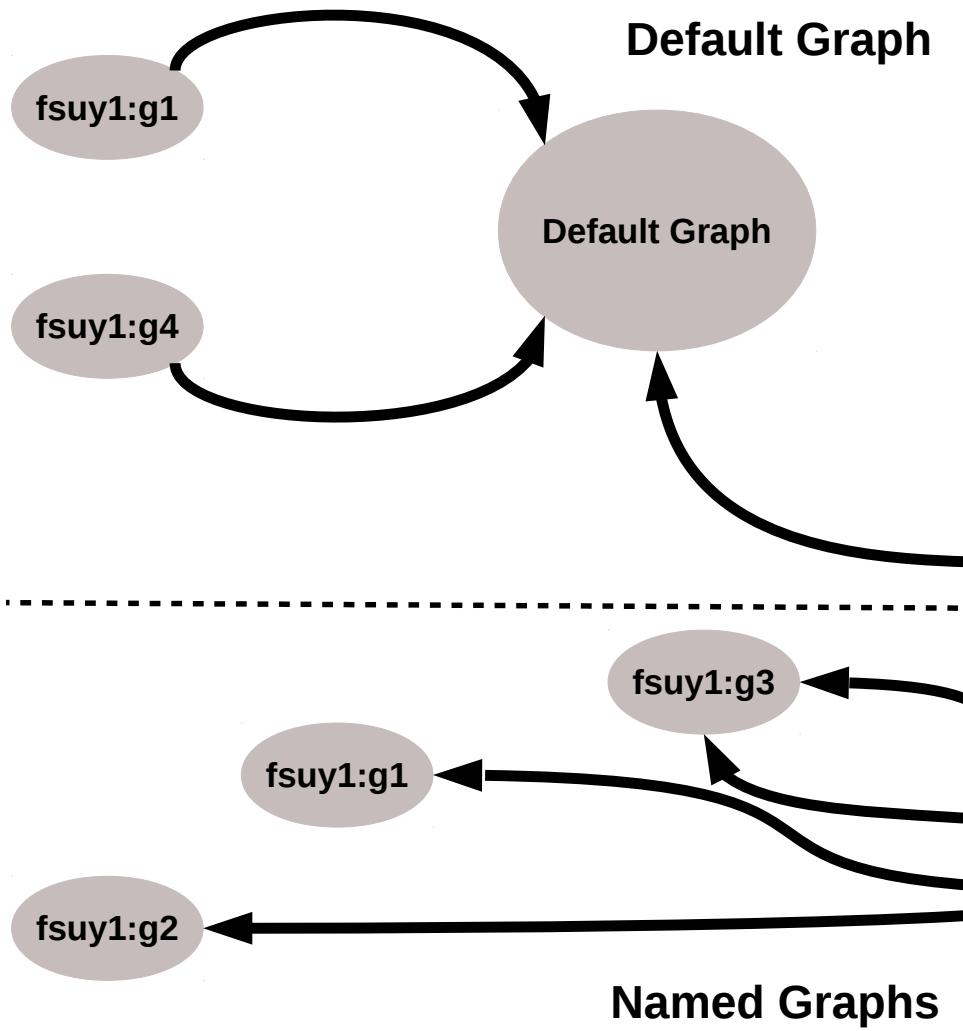
## Query Format

- Queries are executed over an RDF dataset
  - One (or more) default graph
  - zero or more named graphs

- **Named Graphs** can be explicitly addressed via keyword **GRAPH** and the URI of the named graph

```
GRAPH <http://example.org/graph1.rdf>{
        ?x foaf:mbox ?mbox
}
```

# SPARQL


```
PREFIX fsuy1 : …

SELECT …

FROM fsuy1:g1

FROM fsuy1:g4

FROM NAMED fsuy1:g1

FROM NAMED fsuy1:g2

FROM NAMED fsuy1:g3

WHERE {

    … A …

    GRAPH fsuy1:g3{
        … B …
    }

    GRAPH ?G {
        … C …
    }
}
```


# SPARQL

## SPARQL Query Format : Example of Named Graphs

In the previous figure :

- Several "FROM" statements
- Default graphs : facscienceuy1 :g1 and facscienceuy1 :g2
  $\longrightarrow$ Used for each single graph pattern that is not bound to a specified graph
- "NAMED" graphs : facscienceuy1 :g1, facscienceuy1 :g2, facscienceuy1 :g3
  $\longrightarrow$ Used for single graph pattern that is identified via the graph keyword


# SPARQL

## SPARQL Query Format : Example of Named Graphs

```
#Default Graph
#(stored at http://example.org/dft.ttl)

@prefix dc: <http://purl.org/dc/element/1.1/> .
<http://example.org/Donald> dc:publisher "AllegroGraph Apache Lucene Tutorial" .
<http://example.org/Regis> dc:publisher "AllegroGraph Jena TDB Sorl Tutorial" .

#Named Graph: http://example.org/Donald

@prefix foaf: <http://xmlns.com/foaf/0.1> .

_:a foaf:name   "Ronald" .
_:a foaf:mbox   <mailto:ronald@example.com> .

#Named Graph: http://example.org/Regis

@prefix foaf: <http://xmlns.com/foaf/0.1> .

_:a foaf:name   "Regis" .
_:a foaf:mbox   <mailto:regis@example.com> .
```

- The upper one is the default graph with two triples in it
- In the second, we have the foaf with two triples in it


# SPARQL

## SPARQL Query Format : Example of Named Graphs

```
PREFIX foaf: <http://xmlns.com/foaf/0.1>
PREFIX dc: <http://purl.org/dc/element/1.1/>

SELECT ?g?mbox?who
FROM <http://example.org/dft.ttl>
FROM NAMED <http://example.org/Regis>
FROM NAMED <http://example.org/Donald>
WHERE {
    ?g dc:publisher ?swj .
    GRAPH?g {?x foaf:mbox?mbox}
}
```

Default Graph

Named Graph

- Question : use the graph keyword with the variable :
  - Select in the graph the mailboxes and their owner
    $\longrightarrow$ Consider the result as the default graph
  - I have two named graphs

# SPARQL


SPARQL is not only a query Language

- **SPARQL : P**rotocol **a**nd **R**DF **Q**uery **L**anguage :
  - A **Query Language** for RDF Graph Transversal
    *SPARQL Query Language Specification*
  - A **Protocol Layer**, to use SPARQL via HTTP
    *SPARQL Protocol for RDF Specification*
    The communication with the SPARQL endpoint is arrange and defined
  - An **XML Output Format Specification** for SPARQL Queries
    *SPARQL Query XML Results Format*
    Output format for the answer of the query against the SPARQL endpoint


# SPARQL

## SPARQL Result Format

- The result return from a SPARQL endpoint is in form of an XML document

- This document is made up of :
  - The namespace which is a SPARQL namespace
  - The head of the document :
    - contains the name of all the variables that have been used in the query
    - the names of the variables also give the names of the columns of the result table
  - The body of the document :
    - The results are in the $<$ *result* $>$ markup keyword
    - We have a single section for each single result
    - Each variable is bound to some value

# SPARQL

## SPARQL Result Format


- For different types of variables correspond different types of markup language within the result XML file
- For each variable and each result row, you will have one result with the biding to each variable
- For each line, we have a result paragraph within the XML document and within each result paragraph, we have the binding for each variable


# SPARQL

## SPARQL Result Format

- Results are given as well formed and valid XML documents

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
        ...
<sparql>
```

- In a $<$ *head* $>$ element, all variables of the SPARQL query are listed

```
<head>
        <variable name="x" />
        <variable name="hpage" />
        <variable name="name" />
        <variable name="mbox" />
        <variable name="blurb" />
<head>
```


# SPARQL

## SPARQL Result Format

- Within a $<$ *binding* $>$ element a $<$ *head* $>$ variable is bound to a result

```
<result>

        <binding name="x">
                <bnode>r2</bnode>
        </binding>

        <binding name="hpage">
                <uri>http://facscience-uy1.uninet.cm/Donald</uri>
        </binding>

        <binding name="name">
                <literal xml:lang="en">Donald</literal>
        </binding>

        <binding name="age">
                <literal datatype="http://www.w3.org/2001/XMLSchema#integer">22</literal>
        </binding>

        <binding name="mbox">
                <uri>mailto:donald@facscience-uy1.uninet.cm</uri>
        </binding>

</result>
```


# SPARQL

## SPARQL Query Format (Continued)

In addition to SELECT queries, SPARQL allows :

- **ASK**
  - Check if some result exist
  - Check whether there is at least one result
  - Result : true or false
  - Result is delivered as XML or JSON

  **CONSTRUCT**
  - Result : an RDF graph constructed from a template
  - Template : graph pattern with variables from the query pattern
  - Result in RDF/XML or Turtle

- **DESCRIBE**
  - Result : an RDF graph with data about resources
  - Result is RDF/XML or Turtle


# SPARQL

## SPARQL Query Format : CONSTRUCT

CONSTRUCT :

- defines a template for the construction of new RDF Graphs from the SPARQL endpoint result $\longrightarrow$ vocabulary transcription/vocabulary transformation
- Works : "Translate the following knowledge base to the foaf namespace". In the foaf namespace, we also have "name". We have to substitute the employee name with the foaf name to have a foaf output.

```
@prefix org: <http://example.org/ns#>

_:a org:employeeName     "Maxim" .
_:a org:employeeId       "12457" .

_:b org:employeeName     "Donald" .
_:b org:employeeId       "542457" .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1>
PREFIX org: <http://example.org/ns#>

CONSTRUCT {?x foaf:name ?name}
WHERE {?x org:employeeName ?name}
```


# SPARQL

## SPARQL Query Format : CONSTRUCT

## Result as serialized RDF/XML

```
<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:foaf="http://xmlns.com/foaf/0.1">
        <rdf:Description>
                <foaf:name> Ronald </foaf:name>
        </rdf:Description>
        <rdf:Description>
                <foaf:name> Maxim </foaf:name>
        </rdf:Description>
</rdf:RDF>
```

# SPARQL

## SPARQL Protocol


- Method to query/respond of SPARQL queries via HTTP

- A SPARQL URI consists out of 3 parts :
  1. URL of a SPARQL endpoint
     (e.g. http ://example.org/sparql)
  2. RDF Graph(s) to be queried (optional, part of the query string)
     named-graph-uri = http ://example.org/testrdf.rdf)
  3. Query string
     (part of the query string, e.g. query=SELECT...)

- SPARQL protocol is based on HTTP Web protocol

# SPARQL

## SPARQL Protocol


- SPARQL protocol is based on HTTP Web protocol
- To make a query to a SPARQL endpoint :
  - We address of the endpoint
  - Named all the graphs to query
  - The query string
- The full HTTP string involved :
  - The base part of the URI which is the SPARQL endpoint
  - After the question mark comes parameters (the named graph that you address, the next parameters is the SPARQL query expression)


# SPARQL

## SPARQL Protocol : Example

- Simple SPARQL Query

```
PREFIX dc: <http://purl.org/dc/element/1.1/> .
SELECT ?book ?who
WHERE {?book dc:creator ?who}
```

- HTTP Trace of the SPARQL Query

```
GET /sparql/?query=EncodedQuery&default-graph-uri=http://www.other.example/
    books HTTP/1.1
Host: www.other.example
User-agent: my-sparql-client/0.1
```

- Example of the Select query sent within the HTTP protocol
  - The GET request to a specific SPARQL endpoint
  - The entire query
  - The host name
  - The host where the SPARQL endpoint is located
  - The user-agent : the program called is my-sparql-client/0.1

# SPARQL

## SPARQL Protocol : Example


- The HTTP response is composed of :
  - 200 OK - everything is OK
  - Date
  - Name of the server where the SPARQL endpoint is located
  - The content type which is here application
  - The XML document


# SPARQL

## SPARQL Protocol : Example

## HTTP Trace of the SPARQL Response

```
HTTP/1.1 200 OK
Date: Fri, 26 May 2021 20:55:12 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.4 DAV/1.0.3
Connection: close
Content-Type: application/sparql-results+xml
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
        <head>
                <variable name="book" />
                <variable name="who" />
        </head>
        <results ordered="false" distinct="false">
                <result>
                        <binding name="who">
                                <literal>Semantic Search</literal>
                        </binding>
                        <binding name="who">
                                <literal>Semantic Music</literal>
                        </binding >...
                </result>
        </results>
</sparql>
```


# SPARQL

## Popular SPARQL Endpoints

- General purpose SPARQL Endpoint
  `http://sparql.org/sparql.html`
- DBpedia SPARQL Endpoint
  `http://dbpedia.org/sparql`
- DBLP (Computer Science Bibliography) SPARQL Endpoint
  `http://www4wiwiss.fu-berlin.de/dblp/sparql`
- Linked Movie Database SPARQL Endpoint
  `http://linkedmdb.org/sparql`
- CIA World Factbook SPARQL Endpoint
  `http://www4wiwiss.fu-berlin.de/factbook/sparql`

# SPARQL

## SPARQL 1.1


- New features for the queries
- Allow assignment of new variables
- Update function : allow SPARQL to be a fully fletch data manipulation language
- Enables entailment for RDF(S), OWL and RIF : logical inference mechanism, logical connections, logical relations are considered when answering a SPARQL query

# SPARQL

## SPARQL 1.1


- SPARQL query :
  - Assignments (e.g. BIND, SELECT expressions)
  - Aggregate functions (e.g. COUNT, SUM, AVG)
  - Subqueries
  - Negation (EXIST, NOT EXIST, MINUS)
  - Property paths
  - Basic query federation (SERVICE, BINDING)

- Update :
  - **Graph update :** (INSERT DATA, DELETE DATE, INSERT, DELETE, DELETE WHERE, LOAD, CLEAR)
  - **Graph management :** CREATE, DROP, COPY, MOVE, ADD

- Entailment for RDF, RDFS, OWL, RIF
- Service Descriptions


# SPARQL

## SPARQL 1.1 : new variable assignment

Select items and assign an item a new price which is 10% higher than the price which is given in the database.

In the select statement, compute a new price with the variable "prt*1.1" and take as a newly created variable.

In the answer table, there will be the new values for the variables newP computed.


# SPARQL

## SPARQL 1.1 : new variable assignment

```
PREFIX ex: <http://example.org/>
SELECT ?Item (?Pr*1.1 AS ?NewP)
WHERE {?Item ex:price ?Pr}
```

### Data

```
#Data
@prefix ex: <http://example.org/>
ex:lemonade1 ex:price 3 .
ex:beer1     ex:price 3 .
ex:wine1     ex:price 3.50 .
ex:liqueur1  ex:price "n/a" .
```

### Results

| ?Item | ?NewP |
|---|---|
| lemonade1 | 3.3 |
| beer1 | 3.3 |
| wine1 | 3.85 |
| liqueur1 | |

# SPARQL

## Aggregate Functions


- COUNT : counts the number of times the specified value is bound to the given variable
- SUM : Returns the numeric value obtained by summing the values within the aggregate group
- AVG : Calculates the average value for a numeric expression


# SPARQL

## Aggregate Functions

- MIN : Returns the minimum value from the specified set of values
- MAX : Returns the maximum value from the specified set of values
- SAMPLE : Returns an arbitrary value from the specified set of values
  "pick" one element from a group of elements in a non deterministic way.
- GROUP_CONCAT : Performs a string concatenation of all of the values that are bound to the given variable
  It concatenates the values with a designated string separator


# SPARQL

## SPARQL 1.1 : Aggregate Functions

1. Count the number of items in the knowledge base
   Use the aggregate function "Count" to count the number of time the variable item is assign to a value from the database.

2. Count the number of distincts categories in the database

3. Count all items per categories
   Restrict the elements and group them with the keyword GROUP BY

4. Count the items of each categories and consider those which have more than one element.


# SPARQL

## SPARQL 1.1 : Aggregate Functions (example 1)

```
PREFIX ex: <http://example.org/>
SELECT (Count(?Item) AS ?C)
WHERE {?Item ex:price ?Pr}
```

### Data

```
#Data
@prefix ex: <http://example.org/>
ex:lemonade1 ex:price 3 ;
             rdf:type ex:Softdrink .
ex:beer1             ex:price 3 ;
             rdf:type ex:Beer .
ex:wine1             ex:price 3.50 ;
             rdf:type ex:Wine .
ex:wine2             ex:price 4 ;
             rdf:type ex:Wine .
ex:liqueur1          ex:price "n/a" ;
             rdf:type ex:Wine .
```

### Results

| ?C |
|---|
| 5 |


# SPARQL

## SPARQL 1.1 : Aggregate Functions (example 2)

```
PREFIX ex: <http://example.org/>
SELECT (Count(DISTINCT ?T) AS ?C)
WHERE {?Item rdf:type ?T}
```

### Data

```
#Data
@prefix ex: <http://example.org/>
ex:lemonade1 ex:price 3 ;
             rdf:type ex:Softdrink .
ex:beer1             ex:price 3 ;
             rdf:type ex:Beer .
ex:wine1             ex:price 3.50 ;
             rdf:type ex:Wine .
ex:wine2             ex:price 4 ;
             rdf:type ex:Wine .
ex:liqueur1          ex:price "n/a" ;
             rdf:type ex:Wine .
```

### Results

| ?C |
|---|
| 3 |


# SPARQL

## SPARQL 1.1 : Aggregate Functions (example 3)

```
PREFIX ex: <http://example.org/>
SELECT ?T (Count(?Item) AS ?C)
WHERE {?Item rdf:type ?T}
GROUP BY ?T
```

### Data

```
#Data
@prefix ex: <http://example.org/>
ex:lemonade1 ex:price 3 ;
             rdf:type ex:Softdrink .
ex:beer1             ex:price 3 ;
             rdf:type ex:Beer .
ex:wine1             ex:price 3.50 ;
             rdf:type ex:Wine .
ex:wine2             ex:price 4 ;
             rdf:type ex:Wine .
ex:liqueur1          ex:price "n/a" ;
             rdf:type ex:Wine .
```

### Results

| ?T | ?C |
|---|---|
| Softdrink | 1 |
| Beer | 1 |
| Wine | 1 |


# SPARQL

## SPARQL 1.1 : Aggregate Functions (example 4)

```
PREFIX ex: <http://example.org/>
SELECT ?T (Count(?Item) AS ?C)
WHERE {?Item rdf:type ?T}
GROUP BY ?T
HAVING Count (?Item) > 1
```

### Data

```
#Data
@prefix ex: <http://example.org/>
ex:lemonade1 ex:price 3 ;
             rdf:type ex:Softdrink .
ex:beer1             ex:price 3 ;
             rdf:type ex:Beer .
ex:wine1             ex:price 3.50 ;
             rdf:type ex:Wine .
ex:wine2             ex:price 4 ;
             rdf:type ex:Wine .
ex:liqueur1          ex:price "n/a" ;
             rdf:type ex:Wine .
```

### Results

| ?T | ?C |
|---|---|
| Wine | 3 |


# SPARQL

## SPARQL 1.1 : Subqueries

Way to embed SPARQL queries within other queries

- Select books titles of the books that have the same authors than "Jiomekong".

```
SELECT ?T
WHERE {
        ?D foaf:maker ?P; rdfs:label ?T .
        {
                SELECT DISTINCT ?P
                WHERE {
                        ?D foaf:maker <http://dblp.13s.de/~/authors/jiomekong>, ?P .
                        FILTER {?P != <http://dblp.13s.de/~/authors/jiomekong>}
                }
                LIMIT 10
        }
}
```

Result is achieved by first eveluating the inner query


# SPARQL

## SPARQL 1.1 : Negation

Filtering of query solutions is done within a FILTER expression using NOT EXISTS and EXISTS

- Select all persons given that some don't have some attributes (which is the name attribute in our knowledge base)
- Use the FILTER expression "NOT EXIST"

```
PREFIX foaf: <http://xmlns.com/foaf/0.1>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
SELECT ?person
WHERE {
      ?person rdf:type foaf:Person .
      FILTER \textbf{NOT EXISTS} {?person foaf:name ?name}
}
```

Data

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

:regis rdf:type foaf:Person .
:regis foaf:name "Regis" .
:donald rdf:type foaf:Person .
```

Results

| ?Person |
| --- |
| :donald |


# SPARQL

## SPARQL 1.1 : Negation

Filtering of query solutions be removing possible solutions with MINUS

- Minus qualifier works in the same way than "NOT EXIST"
- Select all triples that do not have the given name "Donald"

```
PREFIX : <http://example.org/>
PREFIX foaf: <http://xmlns.com/foaf/0.1>
SELECT DISTINCT ?s
        WHERE {
                ?s ?p ?o .
                MINUS {
                        ?s foaf:givenName "Donald" .
        }
}
```

Data

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

:regis foaf:givenName "Regis" ;
                foaf:familyName "Atemengue" .

:donald foaf:givenName "Donald" ;
                foaf:familyName "Kegne" .

:palma foaf:givenName "Palma" ;
                foaf:familyName "Teko" .
```

Results

| ?s |
|---|
| :regis |
| :palma |


# SPARQL

## Property Paths

- Used to navigate within the RDF graph and formulate property path expression for matching on that graph
- Use to select things that fulfill either one or another condition in the graph
- Used to define sequences of properties :
  - Retrieve people that are know by the people that you know
  - Start with the resources ?x and then, follows the property foaf :knows and for the resources that match the first triples, you also match the foaf :knows and the next foaf :name
- The reverse property is used to reverse the direction of the property
  - For instance, accessing to foaf :mailbox
  - Property can be reverse with the sign


# SPARQL

## Property Paths

- A **Property path** is a possible route through an RDF graph between two graph nodes
  - Trivial case : property path of lenght 1, i.e. a triple pattern
  - **Alternatives :** match one or both possibilities
    { :book1 **dc :title—rdfs :label** ?displayString }
  - **Sequence :** property path of lenght $> 1$
    { ?x foaf :mbox $<$ *mailto* : *regis*@*facscience* − *uy*1.*uninet*.*cm* $>$ .
    ?x **foaf :knows/foaf :knows/foaf :name** ?name . }
  - **Inverse property paths :** reversing the direction of the triple
    { ?x foaf :mbox $<$ *mailto* : *regis*@*facscience* − *uy*1.*uninet*.*cm* $>$ }
    $=$ { $<$ *mailto* : *regis*@*facscience* − *uy*1.*uninet*.*cm* $>$ **foaf :mbox** ?x }