# Semantics, knowledge graphs and ontologies in practice

**Jose Emilio Labra Gayo**
WESO Research group
University of Oviedo, Spain

# Schedule

| Day | Title | Topics |
|---|---|---|
| Day 1. | Semantic Technologies and Knowledge graphs | Semantic Web<br>Linked data<br>Knowledge graphs<br>RDF data model<br>Property graphs<br>Wikibase graphs<br>Examples and applications |
| Day 2. | RDF data modelling and SPARQL | Data modelling exercises with RDF and turtle<br>SPARQL |
| Day 3. | Validating RDF data | Shape Expressions (ShEx)<br>SHACL<br>Validating Knowledge Graphs |
| Day 4. | Advanced topics | ShEx and SHACL compared<br>Reasoning<br>RDFS<br>OWL<br>Nanopublications |

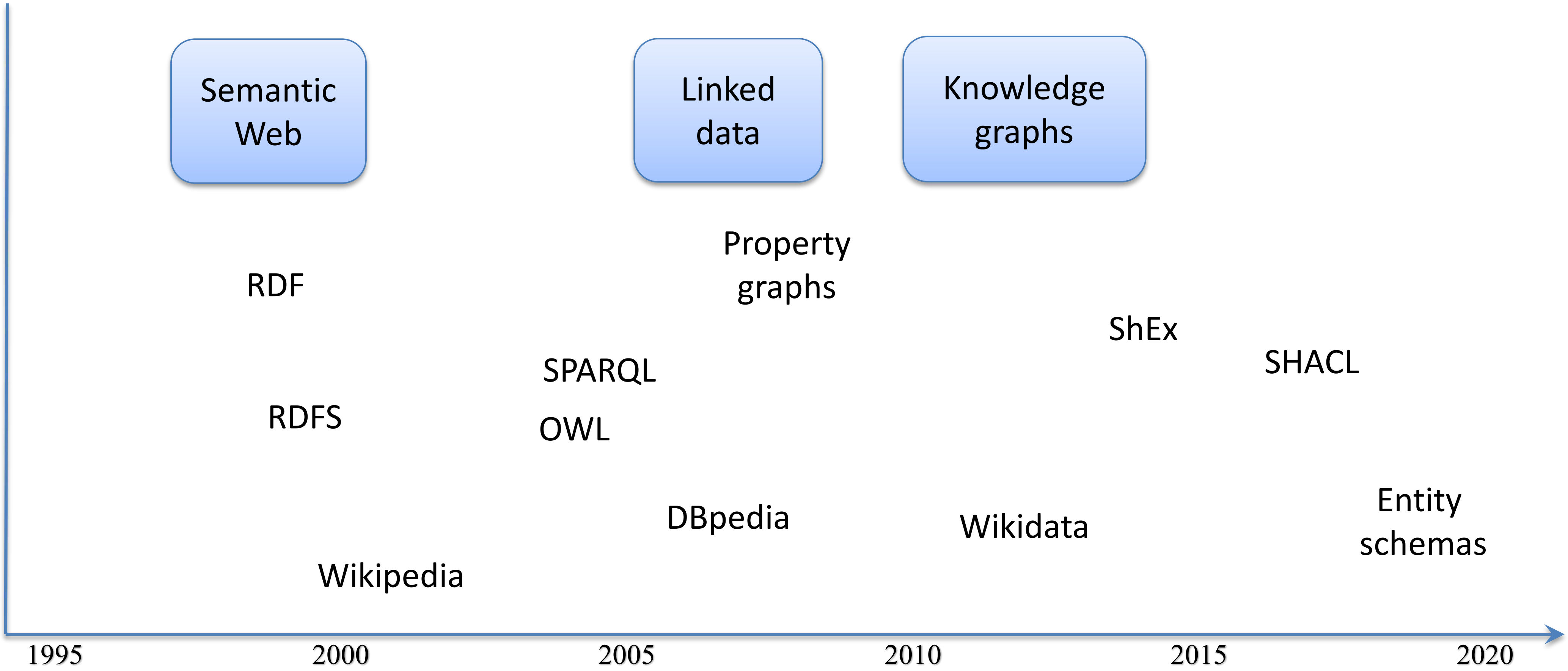
# Session 2. RDF data modelling and SPARQL

**Jose Emilio Labra Gayo**

WESO Research group

University of Oviedo, Spain

# Roadmap



Semantic Web

Linked data

Knowledge graphs

RDF

RDFS

Wikipedia

SPARQL

OWL

Property graphs

DBpedia

Wikidata

ShEx

SHACL

Entity schemas

# SPARQL

Jose Emilio Labra Gayo

Departamento de Informática
Universidad de Oviedo

# SPARQL

SPARQL (April 2006) query language for RDF data

- Similar to SQL, but for RDF
- Based on graph patterns
- It also describes a REST protocol
- SPARQL = SPARQL Protocol And RDF Query Language

SPARQL 1.1 (2013, recommendation)

- Updates, federated queries, etc.

# SPARQL Syntax

- Similar to Turtle
  - URIs between <...>
    - <http://www.example.org/alice>
  - Namespace prefixes as in Turtle
    - `prefix dc: <http://purl.org/dc/terms/>`
    - `dc:creator`
  - Blank nodes
    - `_:node` or between square brackets [ ]
  - Literals between " "
    - `"Alice" "234"^^xsd:integer`
  - Comments start by #
    - `# this is a comment`
- Variables start by ?

# RDF

RDF = Graph model

Different syntaxes: N-Triples, Turtle, RDF/XML

data.ttl

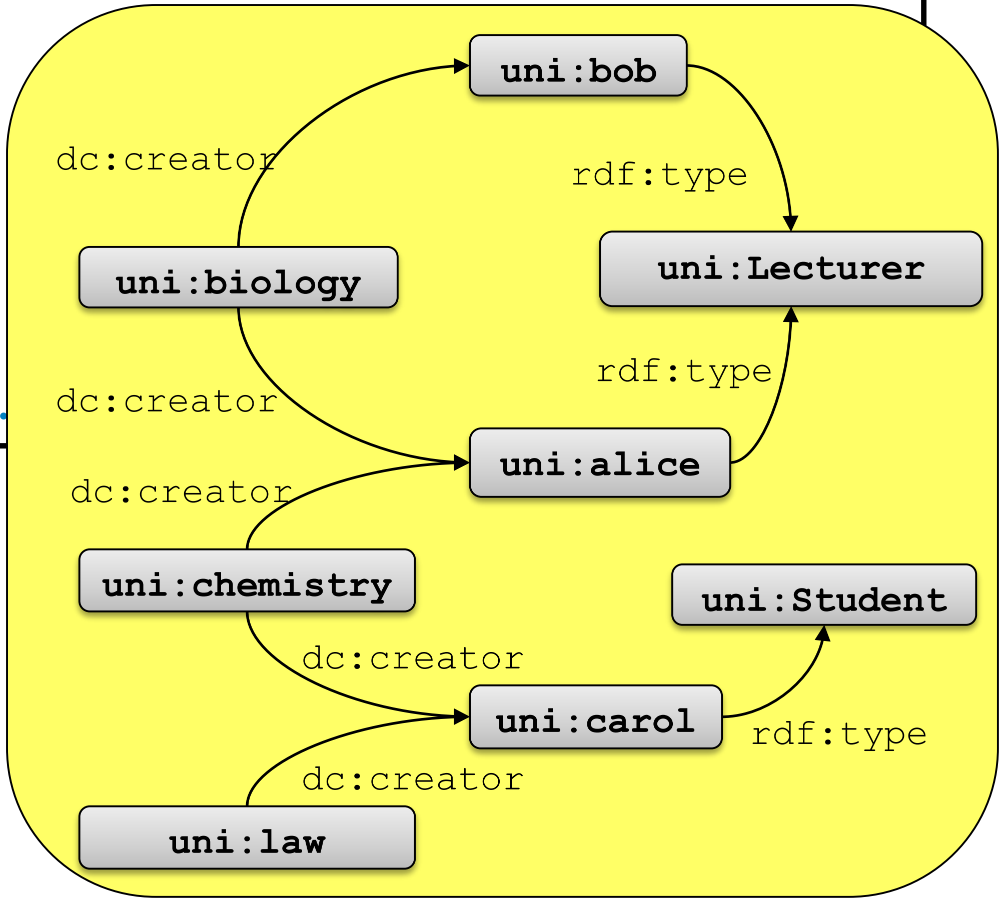
```
@prefix dc:  <http://purl.org/dc/terms/> .
@prefix uni: <http://uniovi.es/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

uni:biology   dc:creator uni:bob .
uni:biology   dc:creator uni:alice .
uni:chemistry dc:creator uni:alice .
uni:chemistry dc:creator uni:carol .
uni:law       dc:creator uni:carol .
uni:alice     rdf:type   uni:Lecturer .
uni:bob       rdf:type   uni:Lecturer .
uni:carol     rdf:type   uni:Student .
```

# RDF graph

RDF data

data.ttl

```
@prefix dc:  <http://purl.org/dc/terms/> .
@prefix uni: <http://uniovi.es/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

uni:biology   dc:creator uni:bob .
uni:biology   dc:creator uni:alice .
uni:chemistry dc:creator uni:alice .
uni:chemistry dc:creator uni:carol .
uni:law       dc:creator uni:carol .
uni:alice     rdf:type   uni:Lecturer
uni:bob       rdf:type   uni:Lecturer
uni:carol     rdf:type   uni:Student .
```



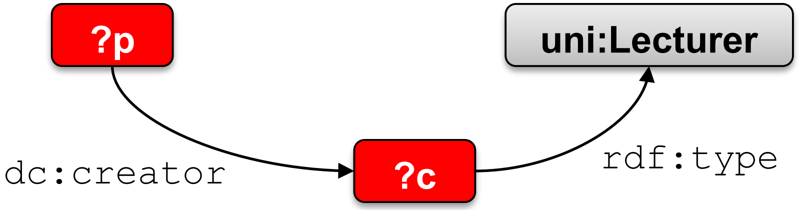
# Simple SPARQL query

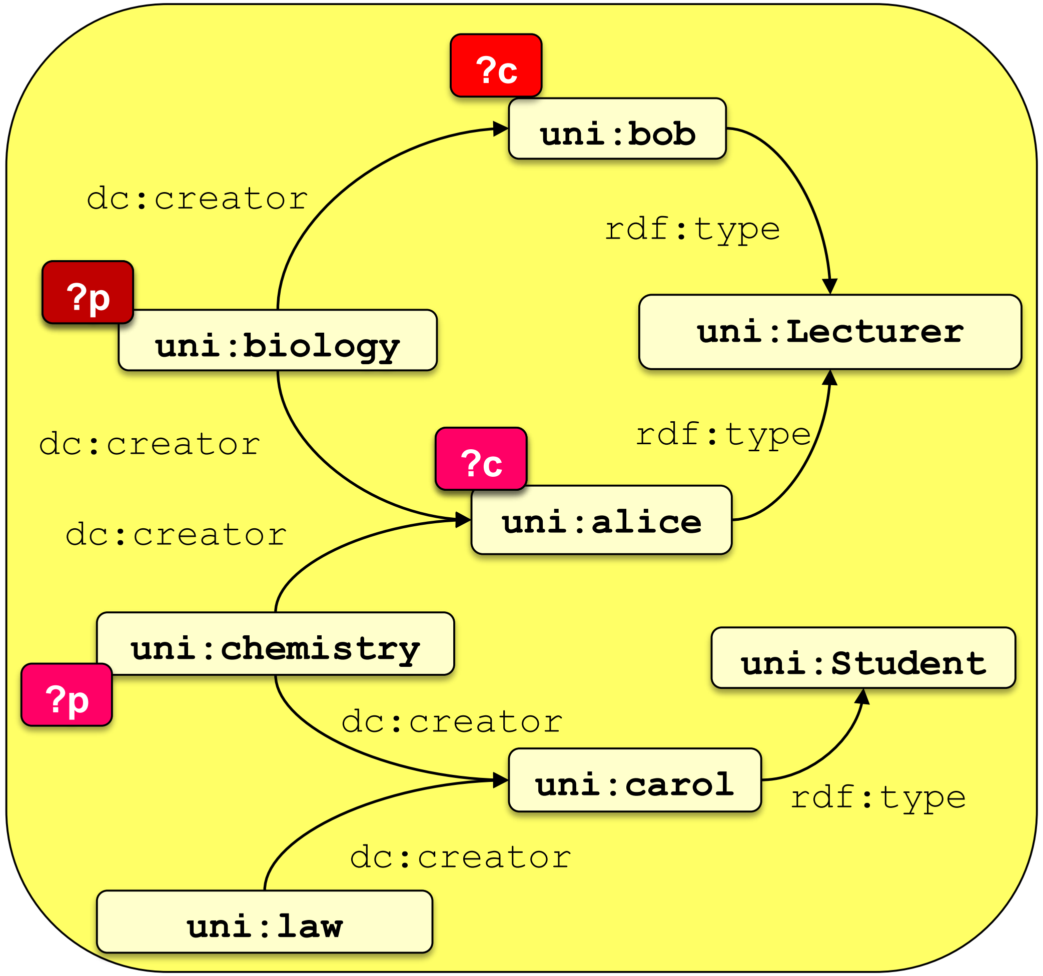
Search resources created by a Lecturer and order them by lecturer

```
prefix dc:  <http://purl.org/dc/terms/>
prefix uni: <http://uniovi.es/>
prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?p ?c WHERE {
 ?p  dc:creator  ?c .
 ?c  rdf:type    uni:Lecturer .
}
ORDER BY ?c
```

# Graph patterns

```
SELECT ?p ?c WHERE {
 ?p  dc:creator  ?c .
 ?c  rdf:type    uni:Lecturer .
}
ORDER BY ?c
```





**Results**

| ?p | ?c |
|---|---|
| uni:biology | uni:alice |
| uni:chemistry | uni:alice |
| uni:biology | uni:bob |

ordered by ?c

Try it: https://goo.gl/fJuUBn

# Playing with SPARQL

Command line tools:

- Apache Jena. https://jena.apache.org/

Online:

- RDFShape: can be used without endpoint
  - http://rdfshape.weso.es/
- YASGUI: can be used to query existing SPARQL endpoints
  - http://yasgui.org/

Creating SPARQL endpoints

- Apache Jena Fuseki: https://jena.apache.org/
- Blazegraph: https://www.blazegraph.com/

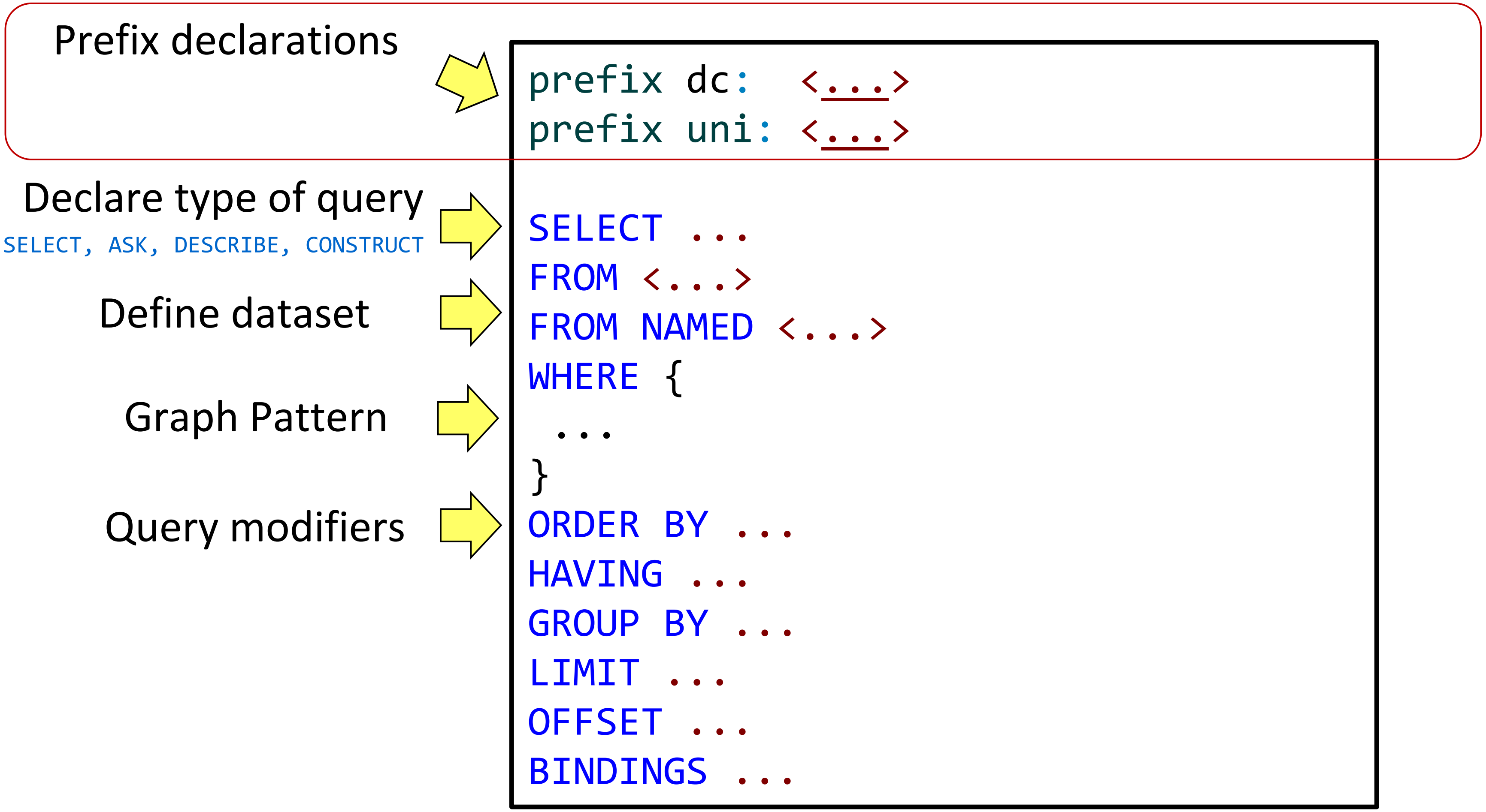
# Some public SPARQL endpoints

| Name | URL | Description |
|---|---|---|
| SPARQLer | http://www.sparql.org/sparql.html | General purpose query endpoint |
| DBpedia | http://dbpedia.org/sparql | RDF data from wikipedia |
| Wikidata | https://query.wikidata.org/ | RDF data from Wikipedia |
| DBLP | http://dblp.rkbexplorer.com/sparql/ | Bibliographic data |
| LinkedMDB | http://data.linkedmdb.org/sparql | Movie database |
| bio2rdf | http://bio2rdf.org/sparql | Linked data for life sciences |

List of SPARQL endpoints:
https://www.w3.org/wiki/SparqlEndpoints

# SPARQL query language

# Parts of a query



Prefix declarations

Declare type of query

SELECT, ASK, DESCRIBE, CONSTRUCT

Define dataset

Graph Pattern

Query modifiers

```
prefix dc:  <...>
prefix uni: <...>

SELECT ...
FROM <...>
FROM NAMED <...>
WHERE {
  ...
}
ORDER BY ...
HAVING ...
GROUP BY ...
LIMIT ...
OFFSET ...
BINDINGS ...
```

# Prefix declarations

Similar to Turtle

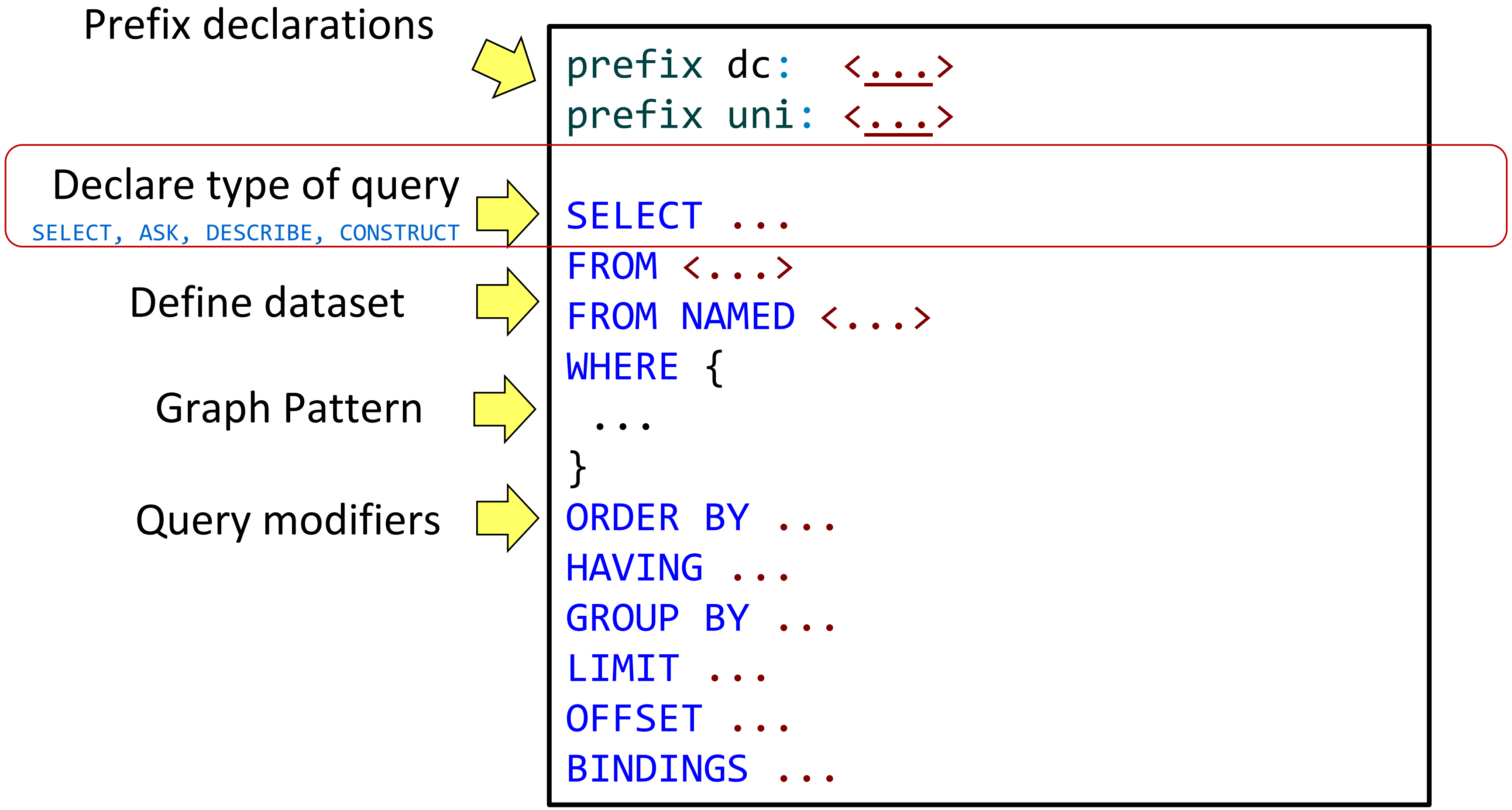
- No need to use @prefix, just prefix
- No need to end prefix declarations by dot

Common aliases used in these slides:

| alias... | stands for... |
|---|---|
| rdf | <http://www.w3.org/1999/02/22-rdf-syntax-ns#> |
| rdfs | <http://www.w3.org/2000/01/rdf-schema#> |
| owl | <http://www.w3.org/2002/07/owl#> |
| xsd | <http://www.w3.org/2001/XMLSchema#> |
| schema | <http://schema.org/> |

Other common prefixes can be found at: http://prefix.cc

# Parts of a query



Prefix declarations

Declare type of query

`SELECT, ASK, DESCRIBE, CONSTRUCT`

Define dataset

Graph Pattern

Query modifiers

```
prefix dc:  <...>
prefix uni: <...>

SELECT ...
FROM <...>
FROM NAMED <...>
WHERE {
  ...
}
ORDER BY ...
HAVING ...
GROUP BY ...
LIMIT ...
OFFSET ...
BINDINGS ...
```

# Types of SPARQL queries

- SELECT return values of variables or expressions
  - Results are a table of values
  - Can have several serializations: XML, JSON
- ASK return true/false
- DESCRIBE return a description of a resource
- CONSTRUCT queries can build RDF triples/graphs

# SELECT queries

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

:alice :name "Alice" ;
       :age 31 .

:bob :name "Robert" ;
     :age 31 .
```

Project out specific variables or expressions

```
SELECT ?n (?age + 1 as ?newAge) WHERE {
   ?x :name ?n ; :age  ?age
}
```

```
--------------------
| n        | newAge |
====================
| "Alice"  | 32     |
| "Robert" | 32     |
--------------------
```

Project out all variables

```
SELECT * WHERE {
   ?x :name ?n ; :age  ?age
}
```

```
-------------------------
| x      | n        | age |
=========================
| :alice | "Alice"  | 31  |
| :bob   | "Robert" | 31  |
-------------------------
```

Project out distinct combinations only

```
SELECT DISTINCT ?age WHERE {
   ?x :name ?n ; :age  ?age
}
```

```
-------
| age |
=======
| 31  |
-------
```

# CONSTRUCT queries

## Construct an RDF result

- Can be used to transform RDF data

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

:alice :name "Alice" ;
       :age   31 .

:bob :name "Robert" ;
     :age   31 .
```

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

CONSTRUCT {
 ?x foaf:name ?name ;
    foaf:age ?age
} where {
 ?x :name ?name ;
    :age  ?age
}
```

Result

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

:alice  foaf:age   31 ;
        foaf:name  "Alice" .

:bob    foaf:age   31 ;
        foaf:name  "Robert" .
```

# ASK queries

ASK return yes or no

Can be used to check errors

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

:alice :name "Alice" ;
       :age  31 .

:bob :name "Robert" ;
     :age  31 .
```

```
PREFIX : <http://example.org/>

ASK WHERE {
 ?x :age  ?age
 FILTER (?age > 18)
}
```

Result

```
Yes
```

# DESCRIBE

Return a description of one or more nodes

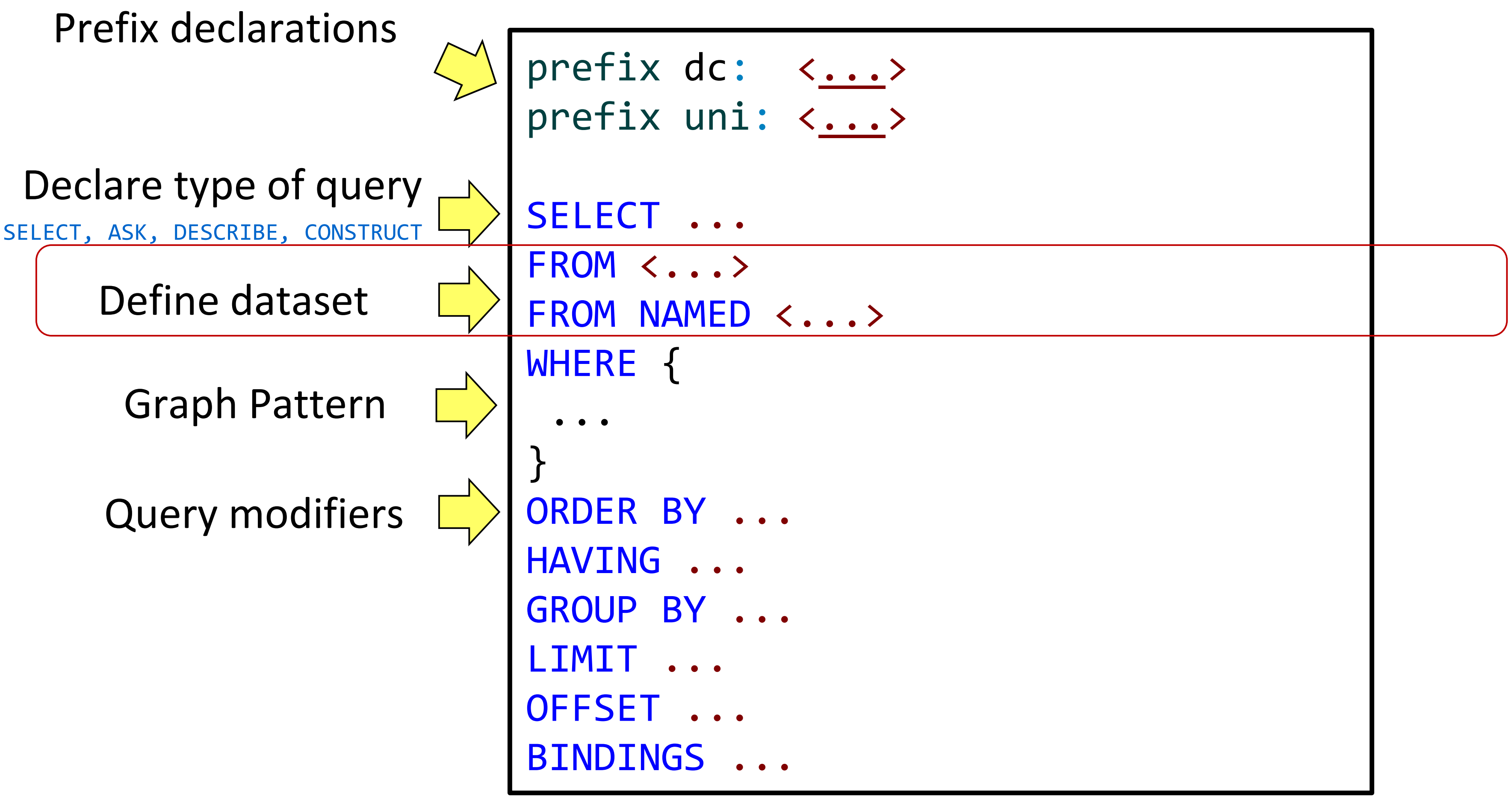
```
@prefix : <http://example.org/>.

:alice :name "Alice" ;
       :age  31 .

:bob :name "Robert" ;
     :age  31 .
```

```
PREFIX : <http://example.org/>

DESCRIBE ?x WHERE {
     ?x :name "Alice" .
}
```

Result

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

:alice  :age     31 ;
        :name    "Alice" .
```

# Parts of a query



Prefix declarations

Declare type of query

SELECT, ASK, DESCRIBE, CONSTRUCT

Define dataset

Graph Pattern

Query modifiers

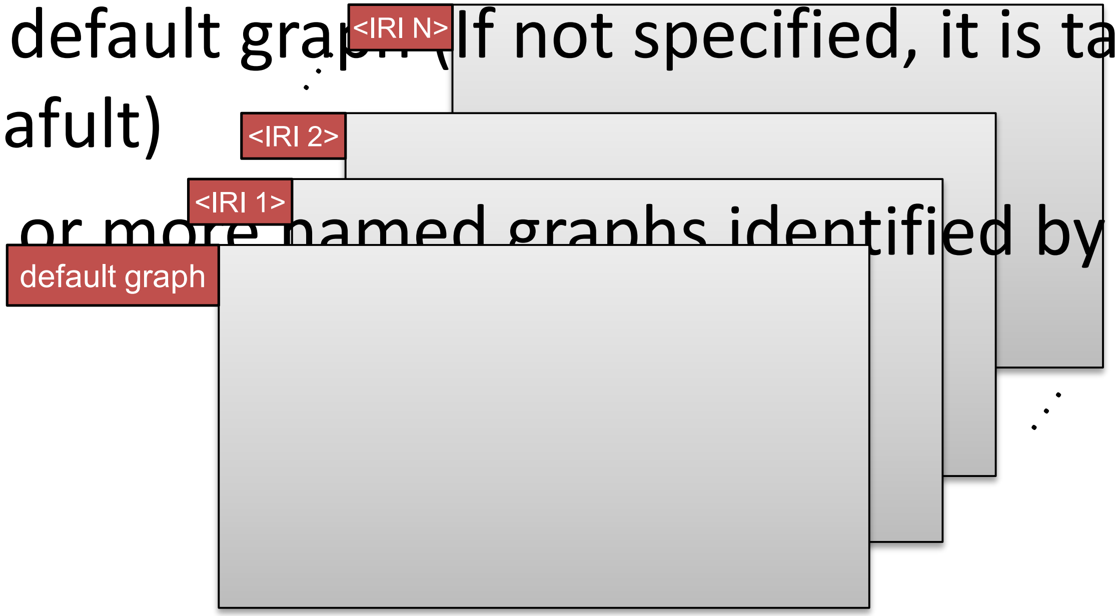
```
prefix dc:  <...>
prefix uni: <...>

SELECT ...
FROM <...>
FROM NAMED <...>
WHERE {
  ...
}
ORDER BY ...
HAVING ...
GROUP BY ...
LIMIT ...
OFFSET ...
BINDINGS ...
```

# RDF datasets

SPARQL queries are executed against an RDF dataset

An RDF dataset has:

- One default graph (If not specified, it is taken by deafult)
- Zero or more named graphs identified by an URI



RDF dataset

# Define dataset using FROM

## FROM declares the URI of the graph to query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?n
FROM <http://www.di.uniovi.es/~labra/labraFoaf.rdf>
WHERE { ?x foaf:name ?n }
```

```
-------------------------------------
| n                                 |
=====================================
| "Jose Manuel Alonso Cienfuegos" |
| "Ivan Herman"                     |
| "Jose Emilio Labra Gayo"          |
-------------------------------------
```

## If several data graphs are declared, they are merged

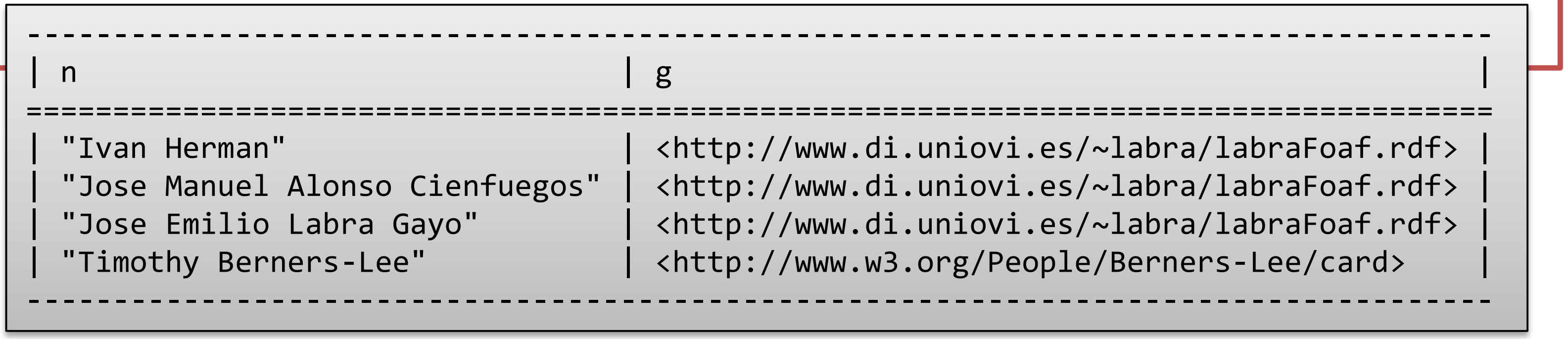
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?n
FROM <http://www.di.uniovi.es/~labra/labraFoaf.rdf>
FROM <http://www.w3.org/People/Berners-Lee/card>
WHERE {
?x foaf:name ?n
}
```

```
-------------------------------------
| n                                 |
=====================================
| "Jose Manuel Alonso Cienfuegos" |
| "Timothy Berners-Lee"             |
| "Ivan Herman"                     |
| "Jose Emilio Labra Gayo"          |
-------------------------------------
```

# Named graphs

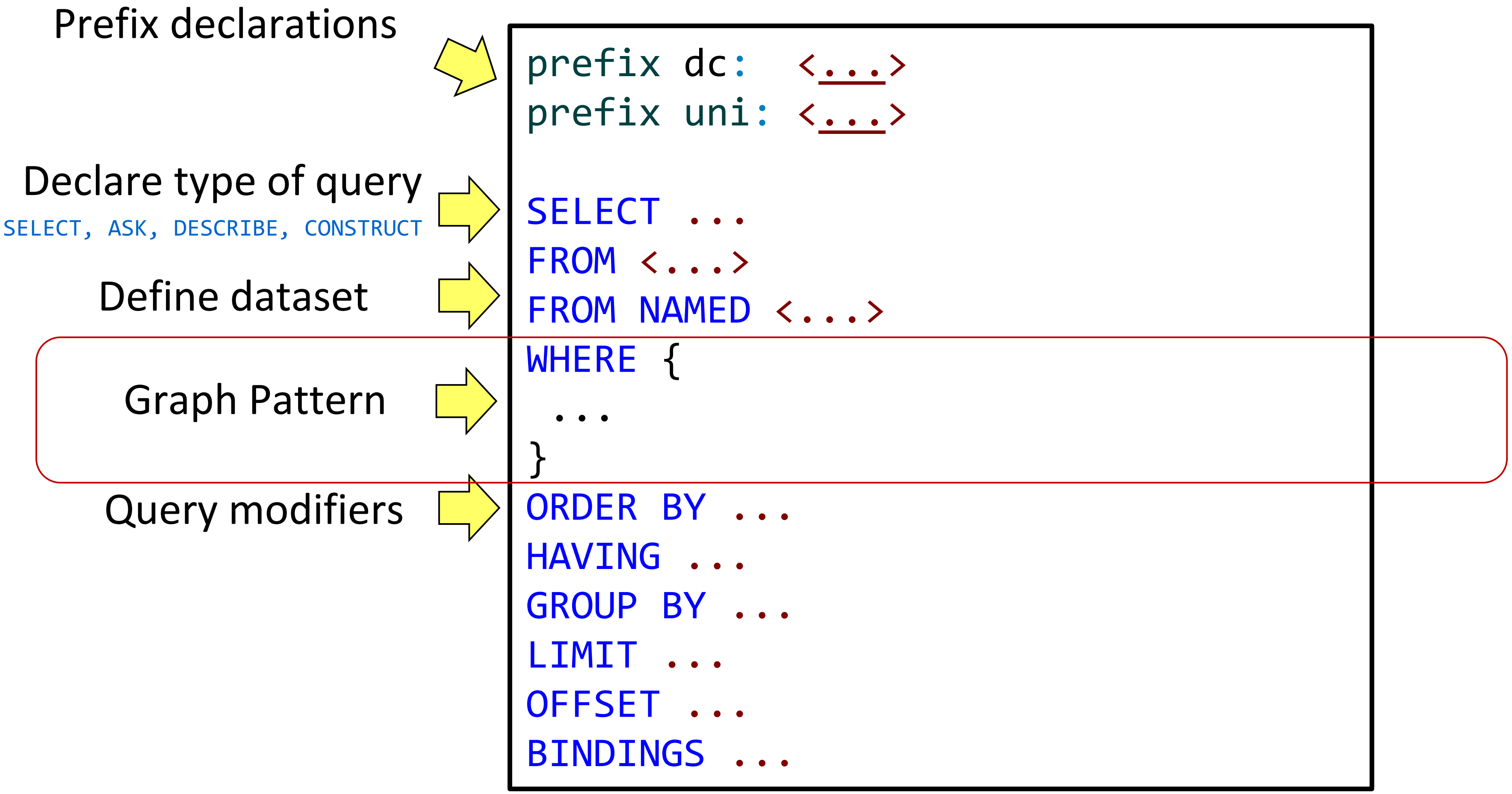
FROM NAMED asigns a name to the input graph

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?n ?g
FROM NAMED <http://www.w3.org/People/Berners-Lee/card>
FROM NAMED <http://www.di.uniovi.es/~labra/labraFoaf.rdf>
WHERE {
 GRAPH ?g { ?x foaf:name ?n }
}
```



```
------------------------------------------------------------------------------------
| n                            | g                                                 |
====================================================================================
| "Ivan Herman"                | <http://www.di.uniovi.es/~labra/labraFoaf.rdf> |
| "Jose Manuel Alonso Cienfuegos" | <http://www.di.uniovi.es/~labra/labraFoaf.rdf> |
| "Jose Emilio Labra Gayo"     | <http://www.di.uniovi.es/~labra/labraFoaf.rdf> |
| "Timothy Berners-Lee"        | <http://www.w3.org/People/Berners-Lee/card>    |
------------------------------------------------------------------------------------
```

# Parts of a query



Prefix declarations

Declare type of query

SELECT, ASK, DESCRIBE, CONSTRUCT

Define dataset

Graph Pattern

Query modifiers

```
prefix dc:  <...>
prefix uni: <...>

SELECT ...
FROM <...>
FROM NAMED <...>
WHERE {
 ...
}
ORDER BY ...
HAVING ...
GROUP BY ...
LIMIT ...
OFFSET ...
BINDINGS ...
```

# Query patterns

Query patterns are made from triple patterns

Triple pattern = RDF triples which can contain variables

Examples of triple patterns

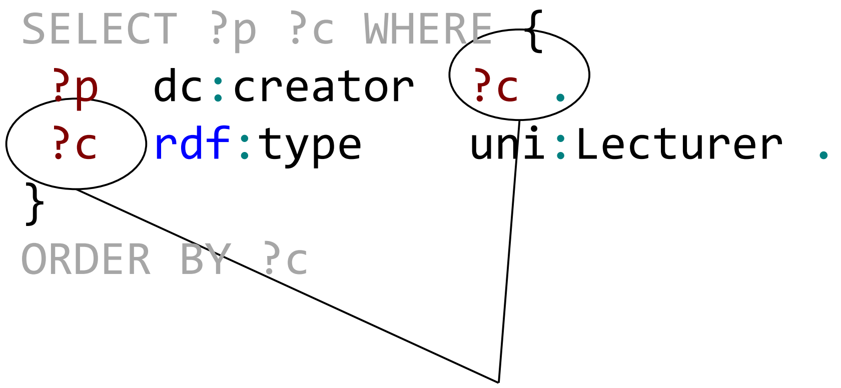
| | |
|---|---|
| `uni:biology dc:creator ?c` | resources that are dc:creator's of uni:biology |
| `?r dc:creator :alice` | resources whose dc:creator is :alice |
| `?r dc:creator ?c` | all resources related by dc:creator property |
| `uni:biology ?p :alice` | properties that relate uni:biology with :alice |
| `?x ?p ?y` | all statements |

# Basic graph patterns

Basic graph pattern = sequence of triple patterns

- The matching process combines the values of variables

Example:



```
SELECT ?p ?c WHERE {
  ?p  dc:creator  ?c .
  ?c  rdf:type    uni:Lecturer .
}
ORDER BY ?c
```

The values of variables must be the same in the results

# Basic graph patterns can have filters

## FILTER limits the set of returned values

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

:alice :name "Alice" .
:alice :age  31 .

:bob :name "Robert" .
:bob :age  12 .

:carol :name "Carol" .
:carol :age  25 .
```

```
PREFIX : <http://example.org/>

SELECT ?n ?e WHERE {
  ?x :name ?n .
  ?x :age  ?e
  FILTER (?e > 18)
}
```

```
| n            | e  |
=====================
| "Carol"      | 25 |
| "Alice"      | 31 |
```

Try it: https://tinyurl.com/y8odenvm

# Filter operators

FILTER uses XPath 2.0 functions and operators

Datatypes: Boolean, Integer, Float, dataTime, etc.

Typical operators: >, <, >=, <=, =, !=, ||, &&

```
PREFIX : <http://example.org/>

SELECT ?n ?e WHERE {
  ?x :name ?n .
  ?x :age  ?e
  FILTER (?e > 30 || ?e < 18)
}
```

Try it: https://tinyurl.com/yavd6bww

# Convert/create datatypes

`str(arg)`: converts its argument to a string

NOTE: URIs must be converted to strings to treat them as such

`datatype(arg)`: returns datatype of a literal

IF `?x = "123"^^xsd:integer`

THEN `datatype(?x) = xsd:integer`

`lang(arg)`: returns the language of a literal

IF `?x = "University"@en`

THEN: `lang(?x) = "en"`

Example

```
prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?label WHERE {
  ?x rdfs:label ?label .
  FILTER (lang(?label) ="en")
}
```

Try it: https://tinyurl.com/y8m3xlyg

# Create datatypes

`uri(arg)`, `iri(arg)`: convert their argument to URI/IRI

`bnode(arg)`: generates a blank node

`strdt(literal,datatype)`: generates a literal with a datatype

```
strdt("123","xsd:integer") = "123"^^<xsd:integer>
```

`strlang(literal,lang)`: generates a literal with some language

```
strlang("University","en") = "University"@"en"
```

# Check datatypes

`isNumeric(arg)` = true if the argument is a number

`isBlank(arg)` = true if the argumento is a blank node

`isLiteral(arg)` = true if the argument is a literal

`isIRI(arg)` = true if the argument is an IRI

# Conditions

`bound(arg)` = `true` if the argument has a value

`exists(pattern)` = `true` if the pattern is satisfied

`not exists(pattern)` = `true` si if the pattern is not satisfied

`if(cond,expr1,expr2)` = if cond = true, returns expr1, otherwise, returns expr2

`coalesce(e1,e2,...)` = returns the first expression that is evaluated without error

# Examples

## Filter numeric values

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

:carol   :age     34 ;
         :name    "Carol" .

:alice   :age     23 ;
         :name    "Alice" .

:bob     :age     "Unknown" ;
         :name    "Robert" .
```

```
PREFIX : <http://example.org/>

SELECT ?age WHERE {
 ?x :age ?age .
 FILTER (isNumeric(?age))
}
```

```
-------
| age |
=======
| 34  |
| 23  |
-------
```

Try it: https://tinyurl.com/yan74uwt

# Functions with strings

`strlen(str)` = length of str

`ucase(str)` converts to uppercase

`lcase(str)` converts to lowercase

`substr(str,start,size?)`= substring from start with some size

```
    substr('camino',3,2)='mi'
```

`strstarts(str1,str2)` = true if `str1` starts with `str2`

`strends(str1,str2)` = true if `str1` ends with `str2`

`contains(str1,str2)` = true if str1 contains str2

`encode_for_uri(str)` = result of encoding str as a uri

`concat(str1,...strN)` = concatenates strings

`langMatches(str,lang)` = true if a string matches some language lang

`regex(str,p,flags)` = true if string matches regular expression p with flags

# Examples with strings

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

:alice   :firstName  "Alice"  ;
         :lastName   "Cooper"  .

:bob     :firstName  "Robert"  ;
         :lastName   "Smith"  .

:carol   :firstName  "Carol"  ;
         :lastName   "King"  .
```

```
PREFIX : <http://example.org/>

SELECT (concat(?firstName,' ',?lastName) AS ?name)
WHERE
{
 ?x :firstName ?firstName .
 ?x :lastName ?lastName .
 FILTER (contains(ucase(?firstName),'A'))
}
```

```
------------------
| name           |
==================
| "Alice Cooper" |
| "Carol King"   |
------------------
```

Try it: https://tinyurl.com/yd3p2l43

# Regex

REGEX invokes regular expression matching

- It is based on XPath 2.0 functions
- `regex(?Expr, ?Pattern [, ?Flags])`
  - `?Expr` = expression to match
  - `?Pattern` = regular expression pattern
  - `?Flags` = matching options

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

:alice   :firstName   "Alice" ;
         :lastName    "Cooper" .

:bob     :firstName   "Robert" ;
         :lastName    "Smith" .

:carol   :firstName   "Carol" ;
         :lastName    "King" .
```

```
PREFIX : <http://example.org/>

SELECT ?firstName WHERE {
 ?x :firstName ?firstName .
 FILTER (regex(?firstName,"^[ABC](.*)"))
}
```

```
-------------
| firstName |
=============
| "Alice"   |
| "Carol"   |
-------------
```

Try it: https://tinyurl.com/yd29xsyz

# Regex

## Regular expressions

^ = start of string

$ = end of string

.  = any character

\d = dígit

? = optional, * = 0 or more, + = 1 or more

X{n} = matches X n times

X{m,n} = matches X from m to n times

**Flags**:

i = ignore case

m = multiple lines

s = simple line

x = removes white spaces

# Numeric functions

abs(n) = absolute value

round(n) = rounds a number n

floor(n) = rounds n down

ceil(n) = rounds n up

rand() = random number between 0 y 1

# Functions with dates

`now()` = returns current instant

`year(i)` = returns the year of some instant i

year("2011-01-10T14:45:13.815-05:00"^^xsd:dateTime) = 2011

`month(i)`, `day(i)`, `hours(i)`, `minutes(i)`, `seconds(i)`, `timezone(i)`, `tz(i)` = similar but return other components

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

:alice  :age    23 ;
        :name   "Alice" .

:bob    :age    20 ;
        :name   "Robert" .
```

```
PREFIX : <http://example.org/>

SELECT (year(now()) - ?age as ?birthYear) WHERE {
  ?x :age ?age .
}
```

```
-------------
| birthYear |
=============
| 1994      |
| 1997      |
-------------
```

# HASH functions

`md5(str)` = applies MD5 algorithm to a str

`sha1(str)`, `sha224(str)`,`sha256(str)`,`sha384(str)`, `sha512(str)` = compute hash of str using the corresponding variations of SHA algorithm

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

:alice :name "Alice" ;
:email "alice@email.com" .

:bob :name "Robert" ;
:email "bob@example.com" .
```

```
PREFIX : <http://example.org/>

SELECT ?name
       (SHA1(?email) AS ?sha1Email)
WHERE {
?x :name ?name .
?x :email ?email .
}
```

```
--------------------------------------------------------------
| name     | sha1Email                                       |
==============================================================
| "Alice"  | "14bf670833d4d7daca31afce1011752c80691459" |
| "Robert" | "a460e37bf4d8e893f8fd39536997d5da8d21eebe" |
--------------------------------------------------------------
```

# Graph union

UNION combines results from several graphs

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

:alice :name "Alice" ;
:age 23 .

:bob :firstName "Robert" ;
:age 20 .
```

```
PREFIX : <http://example.org/>

SELECT ?n WHERE {
 { ?x :name ?n }
 UNION
 { ?y :firstName ?n }
}
```

```
------------
| n        |
============
| "Alice"  |
| "Robert" |
------------
```

# Optional

OPTIONAL allows to define triples which match information if exists, but don't fail if it doesn't exist

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

:alice :name "Alice" ;
       :age 23 .

:bob :name "Robert" .

:carol :name "Carol" ;
       :age 33 .
```

without optional

```
PREFIX : <http://example.org/>

SELECT ?name ?age WHERE {
?x :name ?name .
?x :age ?age
}
```

```
-------------------
| name      | age |
===================
| "Alice"   | 23  |
| "Carol"   | 33  |
-------------------
```

with optional

```
PREFIX : <http://example.org/>

SELECT ?name ?age WHERE {
?x :name ?name
OPTIONAL { ?x :age ?age }
}
```

```
--------------------
| name       | age |
====================
| "Alice"    | 23  |
| "Robert"   |     |
| "Carol"    | 33  |
--------------------
```

# Minus

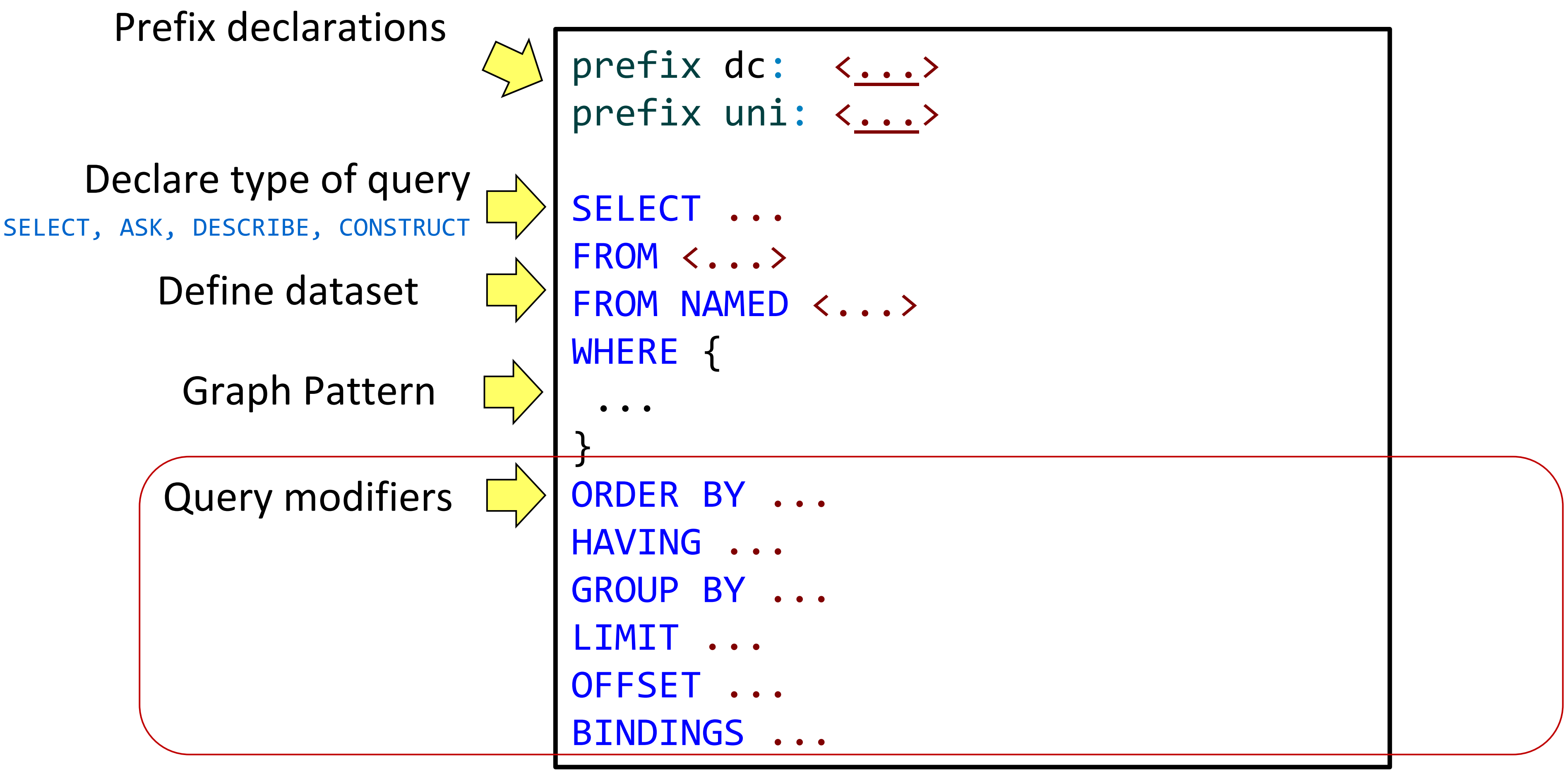
Removes solutions that are compatible with a pattern

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

:alice :name "Alice" ;
       :age 23 .

:bob  :name "Robert" .

:carol :name "Carol" ;
       :age 33 .
```

```
prefix : <http://example.org/>

SELECT ?name WHERE {
 ?x :name ?name
 MINUS {
  ?x :age 33
 }
}
```

```
------------
| name     |
============
| "Alice"  |
| "Robert" |
------------
```

# Parts of a query



Prefix declarations

Declare type of query

SELECT, ASK, DESCRIBE, CONSTRUCT

Define dataset

Graph Pattern

Query modifiers

```
prefix dc:  <...>
prefix uni: <...>

SELECT ...
FROM <...>
FROM NAMED <...>
WHERE {
  ...
}
ORDER BY ...
HAVING ...
GROUP BY ...
LIMIT ...
OFFSET ...
BINDINGS ...
```

# Query modifiers

DISTINCT removes duplicate results

ORDER BY specifies the order of results (ASC, DESC…)

LIMIT n limits the number of results

OFFSET m declares from which result to start

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT DISTINCT ?n
WHERE {
 ?x foaf:knows ?y .
 ?y foaf:name ?n .
}
ORDER BY ASC(?n)
LIMIT 5
OFFSET 2
```

# Bindings

BIND expr AS v = Asigns value of expr to variable v

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

:apples :name "Apples" ;
        :amount 3 ;
        :price 3 .

:oranges :name "Oranges" ;
         :amount 4 ;
         :price 2 .
```

```
prefix : <http://example.org/>

SELECT ?name ?totalPrice
WHERE {
 ?x :name ?name ;
    :amount ?amount ;
    :price ?price .
 BIND ((?amount * ?price) AS ?totalPrice)
}
```

```
---------------------------
| name        | totalPrice |
===========================
| "Oranges" | 8          |
| "Apples"  | 9          |
---------------------------
```

# Bindings in SELECT clause

It is possible to do the binding directly in the SELECT

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

:apples :name "Apples" ;
        :amount 3 ;
        :price 3 .

:oranges :name "Oranges" ;
         :amount 4 ;
         :price 2 .
```

```
prefix : <http://example.org/>

SELECT ?name
       ((?amount * ?price) AS ?totalPrice)
WHERE {
 ?x :name ?name ;
    :amount ?amount ;
    :price ?price .
}
```

```
----------------------------
| name         | totalPrice |
============================
| "Oranges"    | 8          |
| "Apples"     | 9          |
----------------------------
```

# Aggregation functions: AVG, SUM, COUNT, SAMPLE

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

:alice :name "Alice" ;
       :age 23 .

:bob   :name "Robert" ;
       :age 25 .

:carol :name "Carol" ;
       :age 33 .
```

```
PREFIX : <http://example.org/>

SELECT (AVG(?age) AS ?average)
       (SUM(?age) AS ?sum)
       (COUNT(?age) AS ?count)
       (SAMPLE(?age) AS ?sample)
WHERE {
 ?x :age ?age .
}
```

```
-----------------------------------
| average | sum | count | sample |
===================================
| 27.0    | 81  | 3     | 23     |
-----------------------------------
```

# Aggregation functions: MAX, MIN

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

:alice :name "Alice" ;
       :age 23 .

:bob   :name "Robert" ;
       :age 25 .

:carol :name "Carol" ;
       :age 33 .
```

```
PREFIX : <http://example.org/>

SELECT (MAX(?age) AS ?max)
       (MIN(?age) AS ?min)
WHERE {
 ?x :age ?age .
}
```

```
-----------------
| max     | min |
=================
| 33      | 23  |
-----------------
```

# Aggregation functions
# GROUP_CONCAT

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

:alice :name "Alice" ;
       :age 23 .

:bob   :name "Robert" ;
       :age 25 .

:carol :name "Carol" ;
       :age 33 .
```

```
prefix : <http://example.org/>

SELECT (GROUP_CONCAT(?age;
                 SEPARATOR=',')
        as ?ages) where
{
 ?x :age ?age .
}
```

```
--------------
| ages       |
==============
| "23,33,25" |
--------------
```

# Groupings: GROUP_BY

GROUP BY groups sets of results

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

:alice :name "Alice" ;
       :age 23;
       :salary 1200 .

:bob :name "Robert" ;
     :age 25 ;
     :salary 1500 .

:carol :name "Carol" ;
       :age 23 ;
       :salary 2000 .

:dave :name "Dave" ;
      :age 25 ;
      :salary 2500 .
```

```
prefix : <http://example.org/>

SELECT (AVG(?salary) AS ?avgSalary) ?age
WHERE {
?x :age ?age ;
:salary ?salary .
}
GROUP BY ?age
```

```
--------------------
| avgSalary | age |
====================
| 1600.0    | 23  |
| 2000.0    | 25  |
--------------------
```

# Groupings: HAVING

HAVING flters the groups that pass some condition

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

:alice :name "Alice" ;
       :age 23;
       :salary 1200 .

:bob :name "Robert" ;
     :age 25 ;
     :salary 1500 .

:carol :name "Carol" ;
       :age 23 ;
       :salary 2000 .

:dave :name "Dave" ;
      :age 25 ;
      :salary 2500 .
```

```
prefix : <http://example.org/>

SELECT (AVG(?salary) AS ?avgSalary) ?age
WHERE {
 ?x :age ?age ;
    :salary ?salary .
}
GROUP BY ?age
HAVING (?avgSalary > 1800)
```

```
-------------------
| avgSalary | age |
===================
| 2000.0    | 25  |
-------------------
```

# Subqueries

It is possible to define queries inside queries

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

:alice :name "Alice" ;
       :age 23;
       :salary 1200 .

:bob :name "Robert" ;
     :age 25 ;
     :salary 1500 .

:carol :name "Carol" ;
       :age 23 ;
       :salary 2000 .

:dave :name "Dave" ;
      :age 25 ;
      :salary 2500 .
```

```
prefix : <http://example.org/>

SELECT ?name ?salary
       (?salary - ?avgSalary AS ?deviation)
WHERE {
 ?x :name ?name .
 ?x :salary ?salary .
 {
  SELECT (AVG(?salary) AS ?avgSalary) WHERE {
   ?x :salary ?salary .
  }
 }
}
```

```
-----------------------------------
| name     | salary | deviation |
===================================
| "Carol"  | 2000   | 200.0     |
| "Alice"  | 1200   | -600.0    |
| "Dave"   | 2500   | 700.0     |
| "Robert" | 1500   | -300.0    |
-----------------------------------
```

# Property paths

Properties can use a path (similar to regular expressions)

| | |
|---|---|
| p | Match property p |
| (e) | Path grouped in parenthesis |
| ^e | Inverse path e |
| !p | Doesn't match property p |
| e1 / e2 | Path e1 followed by e2 |
| e1 \| e2 | Path e1 or e2 |
| e* | 0 or more e |
| e+ | 1 or more e |
| e? | 0 or 1 e |

# Property paths

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

:alice :name "Alice" ;
        foaf:knows :bob, :carol .

:bob   foaf:name "Robert";
       foaf:knows :carol .

:carol foaf:name "Carol" ;
       foaf:knows :alice .
```

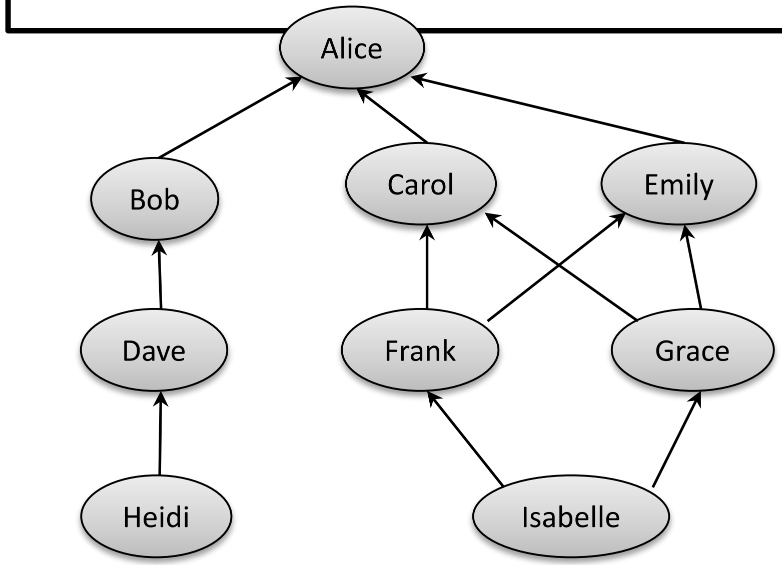
```
prefix : <http://example.org/>
prefix foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name ?friend
WHERE {
?x (foaf:name | :name) ?name ;
  foaf:knows / (foaf:name | :name) ?friend
}
```

```
-----------------------
| name     | friend   |
=======================
| "Alice"  | "Carol"  |
| "Alice"  | "Robert" |
| "Robert" | "Carol"  |
| "Carol"  | "Alice"  |
-----------------------
```

# Property paths

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

:isabelle foaf:knows :frank, :grace .
:frank foaf:knows :carol, :emily .
:grace foaf:knows :carol, :emily .
:carol foaf:knows :alice .
:emily foaf:knows :alice .
:heidi foaf:knows :dave .
:dave foaf:knows :bob .
:bob foaf:knows :alice .
```

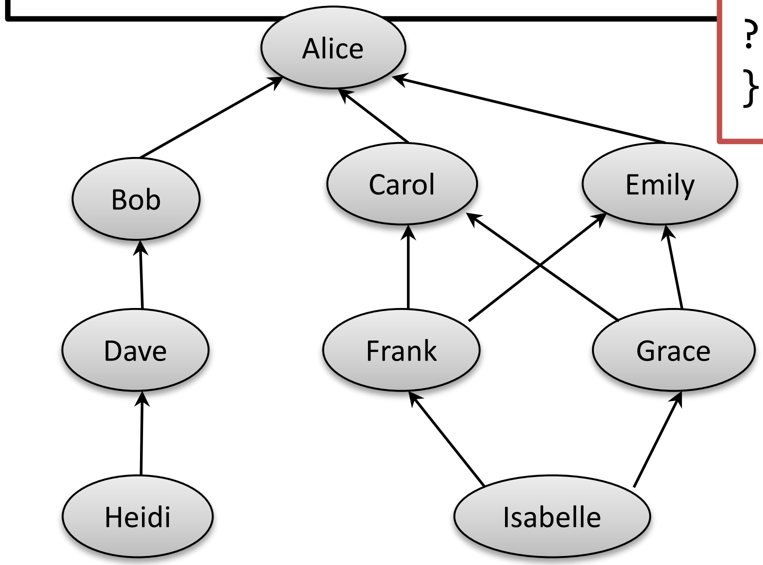
```
prefix : <http://example.org/>
prefix foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?p {
 ?p foaf:knows+ :alice .
}
```



```
-------------
| p         |
=============
| :carol    |
| :grace    |
| :isabelle |
| :frank    |
| :emily    |
| :bob      |
| :dave     |
| :heidi    |
-------------
```

# Property paths

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

:isabelle foaf:knows :frank, :grace .
:frank foaf:knows :carol, :emily .
:grace foaf:knows :carol, :emily .
:carol foaf:knows :alice .
:emily foaf:knows :alice .
:heidi foaf:knows :dave .
:dave foaf:knows :bob .
:bob foaf:knows :alice .
```

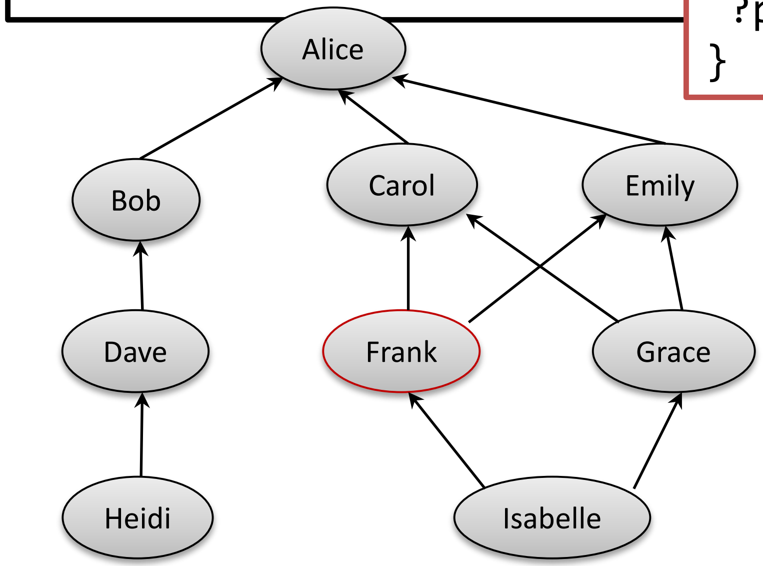


```
prefix : <http://example.org/>
prefix foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?p {
?p foaf:knows/foaf:knows :alice .
}
```

```
----------
| p      |
==========
| :grace |
| :frank |
| :grace |
| :frank |
| :dave  |
----------
```

# Property paths

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

:isabelle foaf:knows :frank, :grace .
:frank foaf:knows :carol, :emily .
:grace foaf:knows :carol, :emily .
:carol foaf:knows :alice .
:emily foaf:knows :alice .
:heidi foaf:knows :dave .
:dave foaf:knows :bob .
:bob foaf:knows :alice .
```



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

SELECT ?p {
 ?p foaf:knows/^foaf:knows :frank .
}
```

```
----------
| p      |
==========
| :grace |
| :frank |
| :grace |
| :frank |
----------
```

# SPARQL Update

# Graph operations

## Update

INSERT DATA = insert triples

DELETE/INSERT… = delete/insert triples conditionally

DELETE DATA = delete triples

LOAD = load triples from a uri

CLEAR = delete all triples from a graph

## Graph management

CREATE = create named graph

DROP = drop graph

COPY…TO… = copy graph

MOVE…TO… = move graph

ADD = insert all elements from a graph in another one

# Insert

INSERT DATA can be used to insert triples

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

INSERT DATA {

:ana  foaf:name "Ana" ;
      foaf:age 18 ;
      :salary 1500 .

 :bob foaf:name "Robert" ;
      foaf:age 20 ;
      :salary 2000 .
}
```

# Insert data in a graph

## INSERT DATA into a named graph

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

INSERT DATA {
 GRAPH <http://example.org/graph1> {
  :alice foaf:name "Alice" ;
         foaf:age 18 ;
         :salary 1500 .
 }
}
```

# Insert

INSERT can insert triples in a graph

- Requires the WHERE clause

```
PREFIX : <http://example.org/>

INSERT {
 ?p :value "GoodSalary".
} WHERE {
 ?p :salary ?salary .
 FILTER (?salary >= 4000)
}
```

# Load a graph

LOAD uri = loads all triples from a graph available at uri

```
LOAD <http://www.di.uniovi.es/~labra/labraFoaf.rdf>
```

# Delete data

DELETE DATA removes all triples in a graph

```
PREFIX : <http://example/org/>

DELETE DATA {
 :alice :age 18 .
}
```

**NOTA**: DELETE DATA does not allow variables

# Delete...where

DELETE WHERE removes triples in a graph specifying a condition

```
PREFIX : <http://example.org/>

DELETE {
 ?x :age ?age .
} WHERE {
 ?x :age ?age .
 FILTER (?age >= 60)
}
```

# Updating information

DELETE/INSERT pattern can be used to update triples in a graph

Example: increment age

```
PREFIX : <http://example.org/>

DELETE { ?x :age ?age }
INSERT { ?x :age ?newAge }
WHERE {
 ?x :age ?age .
 BIND((?age + 1) AS ?newAge)
}
```

# Deleting

CLEAR deletes all triples

It is possible to declare datasets

CLEAR g = Deletes graph g

CLEAR DEFAULT = Deletes default graph

CLEAR ALL = Deletes all graphs

# Universal query

Obtain all triples in all graphs

```
SELECT * WHERE {
 { ?x ?p ?y . }
 UNION
 { GRAPH ?g {
    ?x ?p ?y .
   }
 }
}
```

# Remote services

SERVICE uri = Runs query from a SPARQL

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?name WHERE {
 SERVICE <http://dbpedia.org/sparql> {
  SELECT ?name WHERE {
   ?pais rdf:type dbo:Country .
   ?pais rdfs:label ?name .
   FILTER (lang(?name)='es')
  }
 }
}
```

Some SPARQL endpoints:
http://esw.w3.org/topic/SparqlEndpoints

# Federated queries

Combine results from several endpoints

DBPedia: http://dbpedia.org
IMDB: http://data.linkedmdb.org

```
PREFIX imdb: <http://data.linkedmdb.org/resource/movie/>
PREFIX dcterms: <http://purl.org/dc/terms/>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT * {
 { SERVICE <http://dbpedia.org/sparql> {
    SELECT ?spouseName WHERE {
      ?actor rdfs:label "Javier Bardem"@en ;
      dbo:spouse ?spouse .
      ?spouse rdfs:label ?spouseName .
      FILTER ( lang(?spouseName) = "en" )
    }
  }
 }
 { SERVICE <http://data.linkedmdb.org/sparql> {
    SELECT ?movieName ?movieDate WHERE {
     ?actor imdb:actor_name "Javier Bardem".
     ?movie imdb:actor ?actor ;
            dcterms:title ?movieName ;
            dcterms:date ?movieDate .
    }
   }
 }
}
```

# SPARQL Protocol

# SPARQL Protocol

Defines the actions: query and update and their parameters and their formats

query action

- 2 verbs: GET, POST
- Parameters:
  - `query`: Encoded query
  - `default-graph-uri`: Default graph (optional)
  - `named-graph-uri`: Named graph (optional)

update action

- Only POST with 3 parameters
  - `update`: Update query
  - `using-graph-uri`: Default graph (optional)
  - `using-named-graph-uri`: Named graph (optional)

# Validating RDF using SPARQL?

# Negation by failure pattern in SPARQL

Combining FILTER, OPTIONAL and !BOUND

Example: Search people not married

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

:alice :isMarriedWith :Bob ;
       :name "Alice" .

:bob :isMarriedWith :alice ;
     :name "Robert" .

:carol :name "Carol" .

:dave :isMarriedWith :emily ;
      :name "Dave" .
```

```
PREFIX : <http://example.org/>

SELECT ?n WHERE {
 ?x :name ?n
 OPTIONAL {?x :isMarriedWith ?y }
 FILTER ( !BOUND(?y) )
}
```

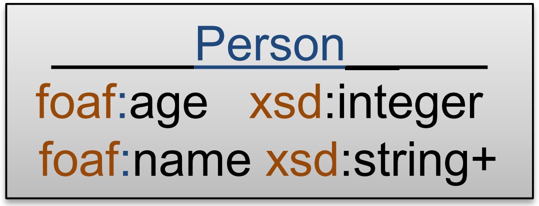
```
-----------
| n       |
===========
| "Carol" |
-----------
```

Does it really return people not married?

# Validating RDF with SPARQL

## Example:

*A person has age (integer) and one or more names (string)*



Person

foaf:age xsd:integer

foaf:name xsd:string+

RDF examples

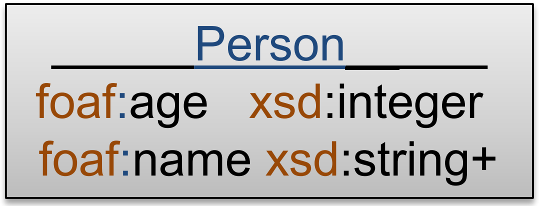
```
:john foaf:age 23;
      foaf:name "John" .

:bob  foaf:age 34;
      foaf:name "Bob", "Robert" .
```

☺

```
:mary foaf:age 50, 65 .
```

☹

# Example of SPARQL query



Person

foaf:age xsd:integer

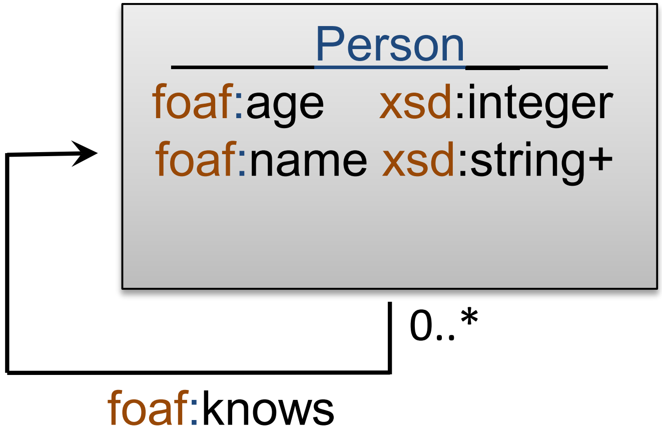
foaf:name xsd:string+

```
ASK {
 { SELECT ?Person {
   ?Person foaf:age ?o .
   } GROUP BY ?Person HAVING (COUNT(*)=1)
 }
 { SELECT ?Person {
   ?Person foaf:age ?o .
   FILTER ( isLiteral(?o) &&
            datatype(?o) = xsd:integer )
   } GROUP BY ?Person HAVING (COUNT(*)=1)
 }
 { SELECT ?Person (COUNT(*) AS ?Person_c0) {
   ?Person foaf:name ?o .
   } GROUP BY ?Person HAVING (COUNT(*)>=1)
 }
 { SELECT ?Person (COUNT(*) AS ?Person_c1) {
    ?Person foaf:name ?o .
    FILTER (isLiteral(?o) &&
            datatype(?o) = xsd:string)
    } GROUP BY ?Person HAVING (COUNT(*)>=1)
 } FILTER (?Person_c0 = ?Person_c1)
}
```

# Is it possible to add recursión to the model?

Example:

*A person has age (integer), one or more names (string) and knows 0 or more values which conform to person*



SPARQL has no recursion

# Validating RDF technologies

ShEx and SHACL can be used to validate RDF

```
<Person> {
  foaf:age     xsd:integer ;
  foaf:name  xsd:string+ ;
  foaf:knows @<Person>
}
```

Example in ShEx (see http://shex.io)

# References

SPARQL by example

SPARQL by example cheatsheet

Learning SPARQL, book by Bob Ducharme

SPARQL 1.1 spec