Class: Is this graph traversal? Why?

```
SELECT ?players
FROM <http://dbpedia.org/data/Giannis_Antetokounmpo>
WHERE {
    dbr:Giannis_Antetokounmpo dbo:draftTeam ?team .
    ?players dbo:team ?team .
    ?players dbo:activeYearsStartYear "2012"^^xsd:gYear .
}
```

Recap: Exercise 7

```
SELECT ?players ?number
FROM <http://dbpedia.org/data/Giannis_Antetokounmpo>
WHERE {
   dbr:Giannis_Antetokounmpo dbo:draftTeam ?team .
   ?players dbo:team ?team .
   ?players dbo:activeYearsStartYear "2012"^^xsd:gYear .
   ?players dbo:number ?number .
}
```

Recap: Exercise 9 - COUNT

What is this query doing? Submit answers to Week 5 – formative

```
SELECT ?team ?p (COUNT (?o) as ?count)

WHERE {
   dbr:Giannis_Antetokounmpo dbo:draftTeam ?team .
   ?team ?p ?o .
}
```

FILTER regex

https://www.w3.org/TR/sparql11-query/#func-regex

https://www.w3.org/TR/xpath-functions/#regex-syntax

FILTER regex(<variable>, <pattern>, <flags>)

Example

FILTER regex(?info, “hello”, “i”) <= case insensitive

Exercise 10 – Filter by regex

- Keep only predicates containing ‘pre’
  (goal here is to keep at least the president)

```
FILTER regex (?p, "pre", "i")
```

- Now keep only those starting with ‘pre’ (need to add a ^)

- What happened?
- **Remove the filter**

Exercise 11 – Grouping and having

Add:

GROUP BY ?team ?p
HAVING (COUNT (?o)>2)

And comment on the results

Exercise 12

Find all information (not properties) on GA’s draft team that contains “buck” irrespective of case

Alter to those starting with “buck”

```
SELECT ?info1 ?info2
FROM <http://dbpedia.org/data/Giannis_Antetokounmpo>
WHERE {
   dbr:Giannis_Antetokounmpo dbo:draftTeam ?team .
   ?team ?info1 ?info2 .
   FILTER regex(?info2, "^buck", "i")
}
LIMIT 100
```

EXISTS / NOT

FILTER EXISTS { triple pattern }

Example:

FILTER EXISTS { ?person foaf:name ?name }

FILTER EXISTS {?team ?p "Milwaukee Bucks"@in .}

Trying it out from prior query (bb)

```
SELECT ?players ?team ?number
WHERE {
    dbr:Giannis_Antetokounmpo dbo:draftTeam ?team .
    ?players dbo:team ?team .
    ?players dbo:activeYearsStartYear "2015"^^xsd:gYear .
    ?players dbo:number ?number .

   FILTER NOT EXISTS {?players dbo:draftTeam dbr:Chicago_Bulls .}

}
```

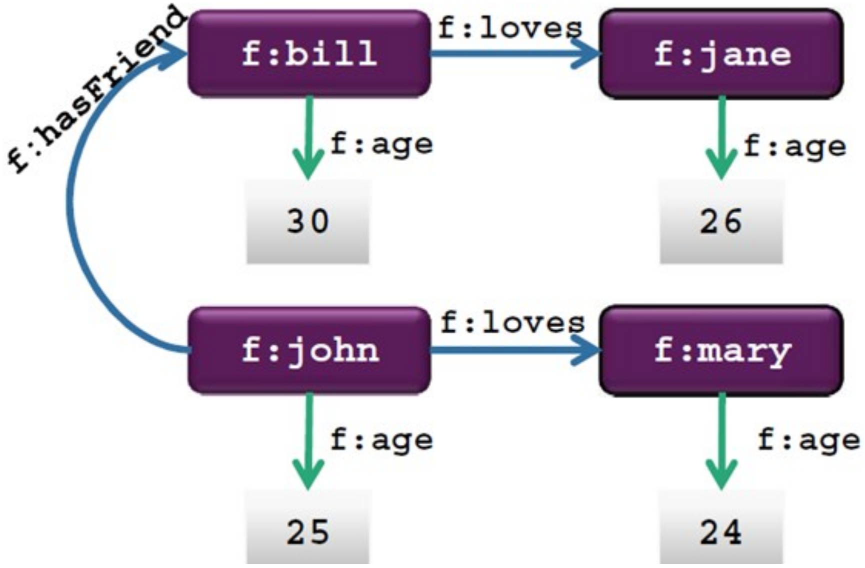
MINUS (=except)

```
…
WHERE { ?s ?p ?o .
            MINUS { ?s foaf:name “George” .}
        }
```

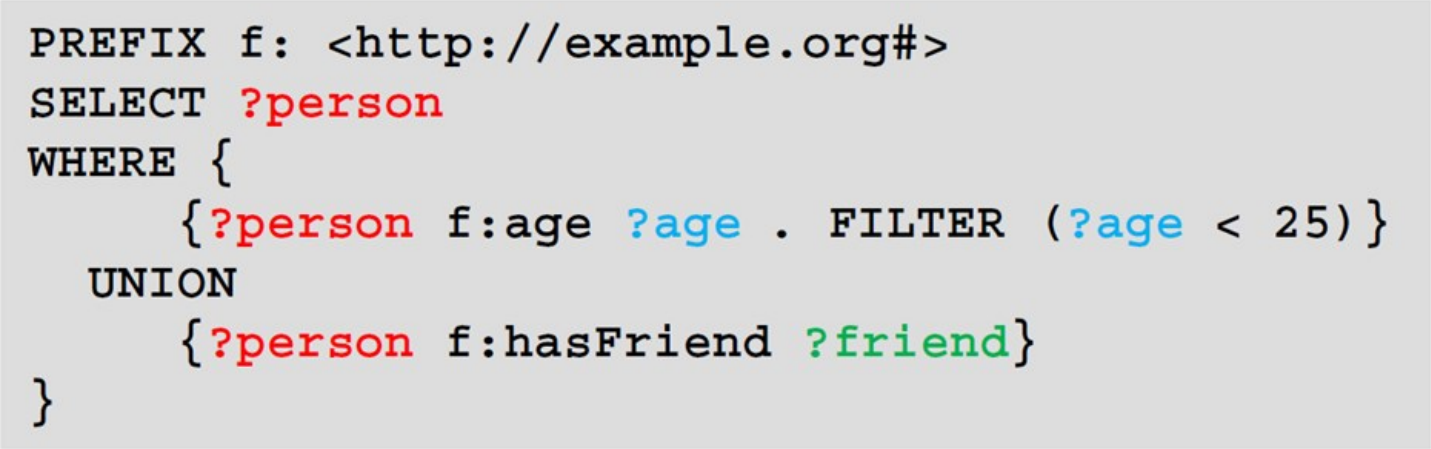
UNION (= or)

Given the data, what is this going to return?

Data



Query



```
PREFIX f: <http://example.org#>
SELECT ?person
WHERE {
      {?person f:age ?age . FILTER (?age < 25)}
  UNION
      {?person f:hasFriend ?friend}
}
```

Exercise 12

Select all players/numbers that currently play for GA’s draft team

That started in 2015
OR
That started in 2012

Discuss in class

Exercise 12

```
SELECT ?players ?team ?number
WHERE {
 {
   dbr:Giannis_Antetokounmpo dbo:draftTeam ?team .
   ?players dbo:team ?team .
   ?players dbo:activeYearsStartYear "2015"^^xsd:gYear .
   ?players dbo:number ?number .
 }
UNION
 {
   dbr:Giannis_Antetokounmpo dbo:draftTeam ?team .
   ?players dbo:team ?team .
   ?players dbo:activeYearsStartYear "2012"^^xsd:gYear .
   ?players dbo:number ?number .
 }
}
```

Exercise 13

Select the subject of all triples in dbpedia

which have a property **dbo:team**
OR
All subjects annotated with the predicate **dbo:league**
and the object **dbr:National_Basketball_Association**

Submit to Weekly activities => Week 5 formative 2

Exercise 13

```
SELECT ?p ?s
WHERE {
  {
?s dbo:league dbr:National_Basketball_Association .
  }
UNION
 {
?p dbo:team ?team .
  }
}
```

Introducing ‘optional’

```
WHERE {
     ?s ?p ?o.
    OPTIONAL { ?s2 ?p2 ?o2 . }
}
```

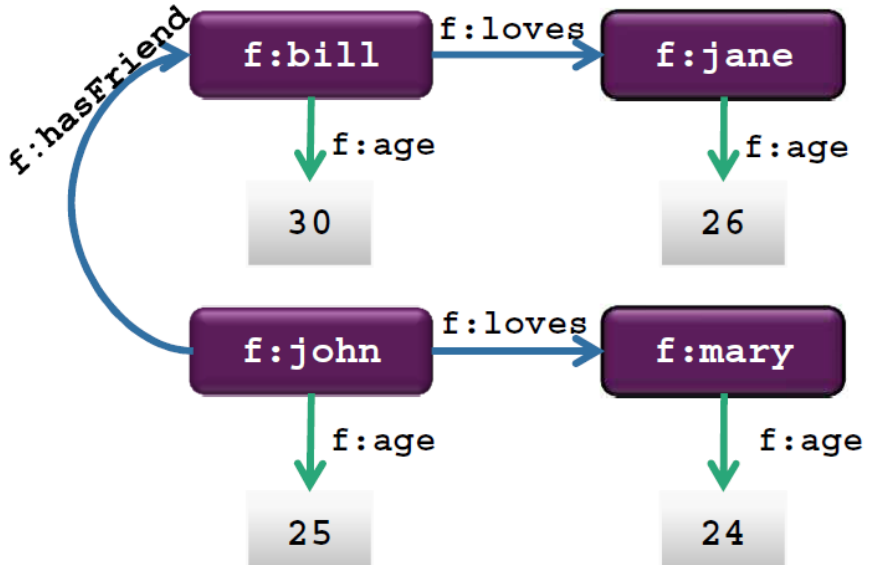
Optional

If you have it, show it!

Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age ?lover
WHERE {
  ?person f:age ?age .
  OPTIONAL {?person f:loves ?lover}
}
```

Data



Optional

If you have it, show it!

Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age ?lover
WHERE {
  ?person f:age ?age .
  OPTIONAL {?person f:loves ?lover}
}
```

Result

person	age	lover
f:bill	30	f:jane
f:john	25	f:mary
f:mary	24	
f:jane	26	

Recap: More formally

Ontology is a formal explicit description of:

- **concepts** in a domain of discourse (a.k.a. **classes**),
- **properties** of each concept describing various features and attributes of the concept (**slots**; sometimes called **roles**),
- and **restrictions** on slots (**facets**; sometimes called role restrictions).

Classes, subclasses, …

- Classes describe concepts in the domain.
  Example,
  - a class of wines represents all wines.
  - Specific wines are instances of this class.

- A class can have **subclasses** that represent concepts that are **more specific** than the superclass.
- For example, we can divide the class of all wines into red, white, and rosé wines. Alternatively, we can divide a class of all wines into sparkling and non-sparkling wines.

THE AMERICAN COLLEGE OF GREECE
Pierce
Deree
Alba
1875

Slots (properties)

- Slots describe properties of classes (**and instances)**
- Château Lafite Rothschild Pauillac wine:
  - has a full body
  - is produced by the Château Lafite Rothschild winery.
- Specifically: slot “body” has a value of “full” and slot “maker” with the value “Château Lafite Rothschild winery”
- **At the class level, we can say that instances of the class Wine will have slots describing their flavor, body, maker of the wine and so on…**

Slots (properties)

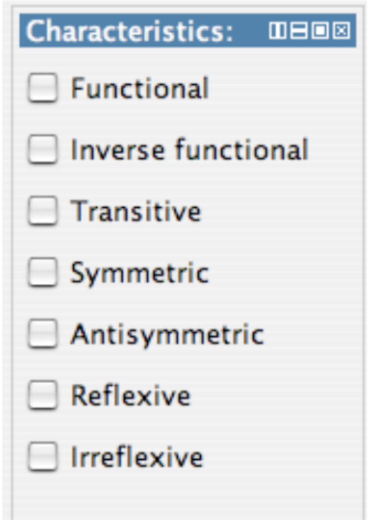
- **Slot cardinality**
- Slot cardinality defines how many values a slot can have. Some systems distinguish only between
  - single cardinality (allowing at most one value) and
  - multiple cardinality (allowing any number of values)
- Examples:
  - A **body** of a wine will be a single cardinality slot (a wine can have only one body).
  - Wines produced by a particular winery fill in a multiple-cardinality slot ‘**produces**’ for a Winery class.

So how do we develop an ontology?

- Developing an ontology includes:
  - defining classes in the ontology,
  - arranging the classes in a taxonomic (subclass–superclass) hierarchy,
  - defining slots and describing allowed values for these slots,
  and
  - filling in the values for slots for instances.

- We can then create a knowledge base by defining individual instances of these classes filling in values and restrictions.

Property Characteristics



Characteristics:

□ Functional

□ Inverse functional

□ Transitive

□ Symmetric

□ Antisymmetric

□ Reflexive

□ Irreflexive

- Functional:
  For a given individual, at most **one** individual that is related to the individual via the property. (*hasBirthMother*)
  = single valued properties or features.

- Transitive:
  If a property is transitive, and the property relates individual a to individual b, and also individual b to individual c, then we can infer that individual a is related to individual c via the property. (*isAncestor*)

- Symmetric: hasSibling; Antisymmetric: cannot be symmetric (flowsFrom); Reflexive: knows (why?); Irreflexive: cannot be reflexive (ideas?)

In-Class Exercise

- Make hasIngredient transitive

1. Select the hasIngredient property in the property hierarchy on the “Object Properties” tab.

2. Tick the “Transitive” tick box on the “Property Characteristics View”.

- Select the isIngredientOf property, which is the inverse of hasIngredient. Ensure that the **transitive** tick box is ticked.
- Make the **hasBase** property **functional => explain why in a comment**

Beware: Domains and Ranges

- It is important to realise that in OWL domains and ranges should not be viewed as constraints to be checked.
- They are used as ‘axioms’ in **reasoning**.
- For example
  - if the property hasTopping has the domain set as Pizza, and
  - we then applied the hasTopping property to individuals that are members of the class IceCream,
  => this would generally not result in an error.
  => Class: any guesses for what to expect?

Beware: Domains and Ranges

- It is important to realise that in OWL domains and ranges should not be viewed as constraints to be checked.
- They are used as ‘axioms’ in **reasoning**.
- For example if the property hasTopping has the domain set as Pizza and we then applied the hasTopping property to individuals that are members of the class IceCream, this would generally not result in an error.
  => It would/could be used to infer that the class IceCream must be subclass of Pizza!