MANCHESTER 1824

The University of Manchester

COMP60411: Modelling Data on the Web

Graphs, RDF, RDFS, SPARQL

Week 5

Bijan Parsia & Uli Sattler

University of Manchester

Feedback on SE3

In 200-300 words, explain […] In particular, explain which style of query is the "most robust" in the face of such format changes.

(As usual, if you are unsure whether you understand the exact meaning of a term, e.g., 'robust', you should look it up.)

Wikipedia: In computer science, robustness is the ability of a computer system to cope with errors during execution. …

- only few discussed robustness!
  - many mentioned which style requires which changes
  - but few discussed how that affects
    - **likelihood** of errors
    - which **kind** of errors (silent/breaking totally)
- many confused **format** with **schema**
  - but they are different concepts!

Feedback on SE3

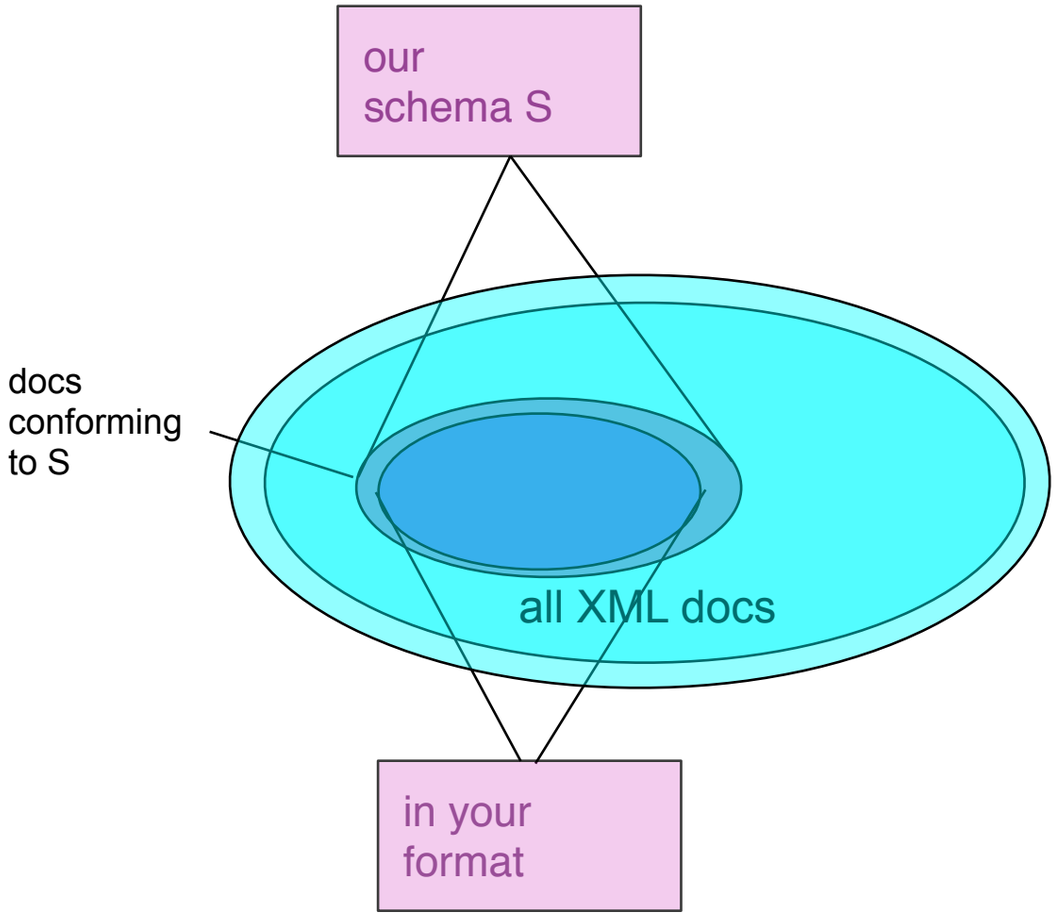
- mostly better :)
  - I see clear improvements in most students!
- an XPath expression **is** an XQuery query
- some still *make things up*:
  - "X is mostly used for Y"
  - "X is better for efficiency than Y"
  - "Using X makes processing faster"
  - …statements like this require evidence/reference: "According to [3], X is mostly used for Y".
- consider your situations carefully:
  - do we need to update schema?
    - if yes, …
    - if no,…

Formats for ExtRep of data (SE4)

- a **format** (e.g., for occupancy of houses) consists of
  1. a data structure formalism (csv, table, XML, JSON,…)
  2. a conceptual model, independent of [1]
  3. **schema(s)** formalising/describing the format
     - documents describing (some aspects of our) design
     - e.g., occupancy.rnc, occupancy.sch,…
  4. the set of **(XML) documents** conforming to a format
     - concrete *embodiments* of our design
     - e.g., an XML document describing Smiths, HighBrow, …
- [2&3] the CM & schema can be
  - explicit/tangible or implicit
    - written down in a note versus ‘in our head’ or by example
  - formalised or unformalised
    - ER-Diagram, XSD versus drawing, description in English
- [4] the documents are implicit

Formats for ExtRep of data (SE4)

e.g., XML-based



Formats for ExtRep of data (SE4)

- Consider 2 formats $F_1 = <DS_1, CM_1, S_1, D_1>$
  $F_2 = <DS_2, CM_2, S_2, D_2>$
- it may be that
  - S_1 only captures some aspects of D_1
  - S_1 is only a description in English
  - $D_1 = D_2$ but $S_1 \neq S_2$
  - $DS_1 = DS_2$ and $CM_1 = CM_2$ but $S_1 \neq S_2$ and $D_1 \neq D_2$
  - …and that F_1 makes better use of DS_1's features than DS_2

- When you design a **format**, you design each of its aspect and
  – how much you make explicit
  – how you formalise CM, S

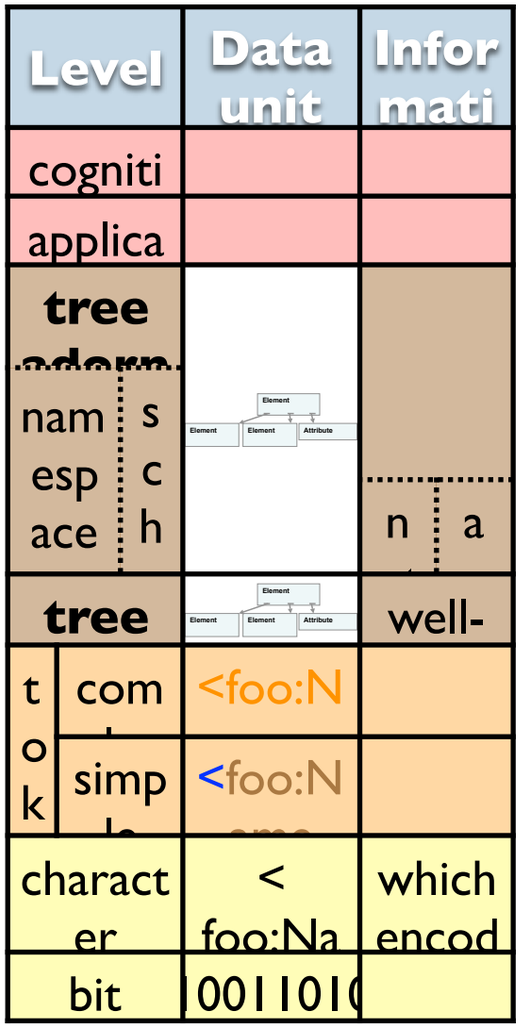
Today

- General concepts: recap of
  - data models
  - pain points
  - formats
  - error handling
  - schemas,…
- New data model & technologies: graph-based DM
  - RDF
  - RDFS, a schema language for RDF
    - but quite different from all other schema languages
  - SPARQL, a data manipulation mechanism for RDF
- Retrospective session

Re-cap of Data Models

Recall: core concepts

- We look at **data models,**
  - shape: none, tables, trees, graphs,…
- and **data structure formalisms** for the above
  - [tables] csv files, SQL tables
  - [trees] sets of feature-value pairs, XML, JSON
  - [graphs] RDF
- and **schema languages** for the above
  - [SQL tables] SQL
  - [XML] RelaxNG, XSD, Schematron,…
  - [JSON] JSON Schema
- and **manipulation mechanisms**
  - [SQL tables] SQL
  - [XML] DOM, SAX, XQuery,…
  - [JSON] JSON API,…



Recall: core concepts

- Each Data Model was **motivated** by
  - representational **needs** of some **domain** and
  - **pain** points
    - **Fundamental** Pain Points
      - Mismatch between the domain and the data structure
    - **Tech-specific** Pain Points
      - XPath Limitations
- **Alleviating** pain
  - Try to squish it in
    - E.g., encoding trees in SQL
    - E.g., layering
  - Polyglot persistence
    - Use multiple data models

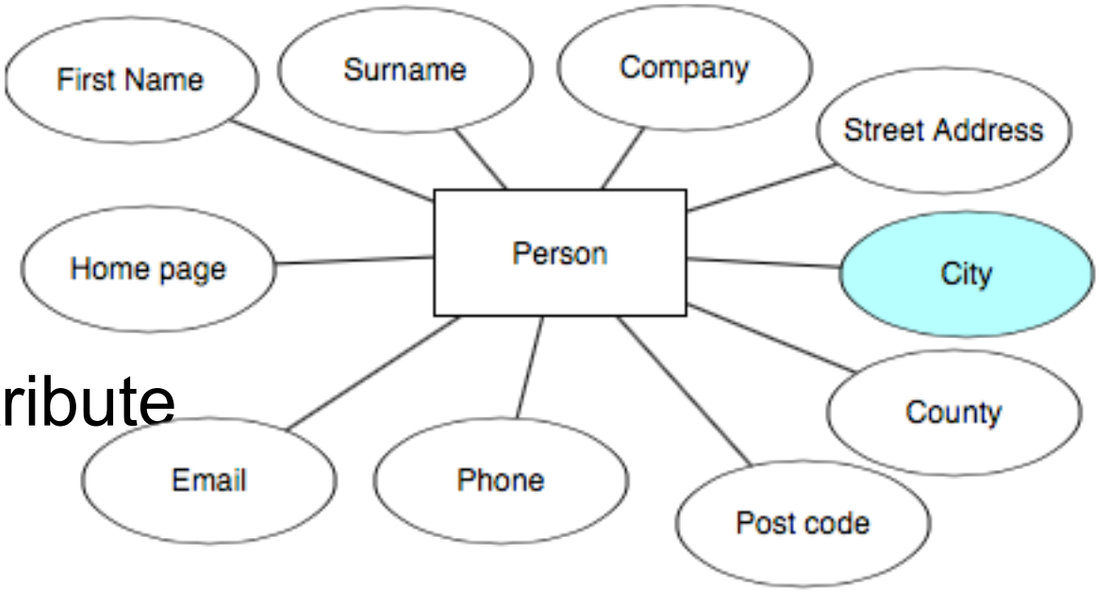
It's important to understand the
- pain points &
- trade offs

Domains/applications discussed so far

- People, addresses, personal data
  - with(out) management structure
- SwissProt protein data
- Cartoons
- Arithmetic expressions
  - [CW1] easy, binary expressions with students, attempts, etc.
  - [CW2, CW3] nested expressions of varying parity
- Horse sharing
  - as an example for ‘sharing’ applications
  - e.g., AirBnB, MoBike, ride shares

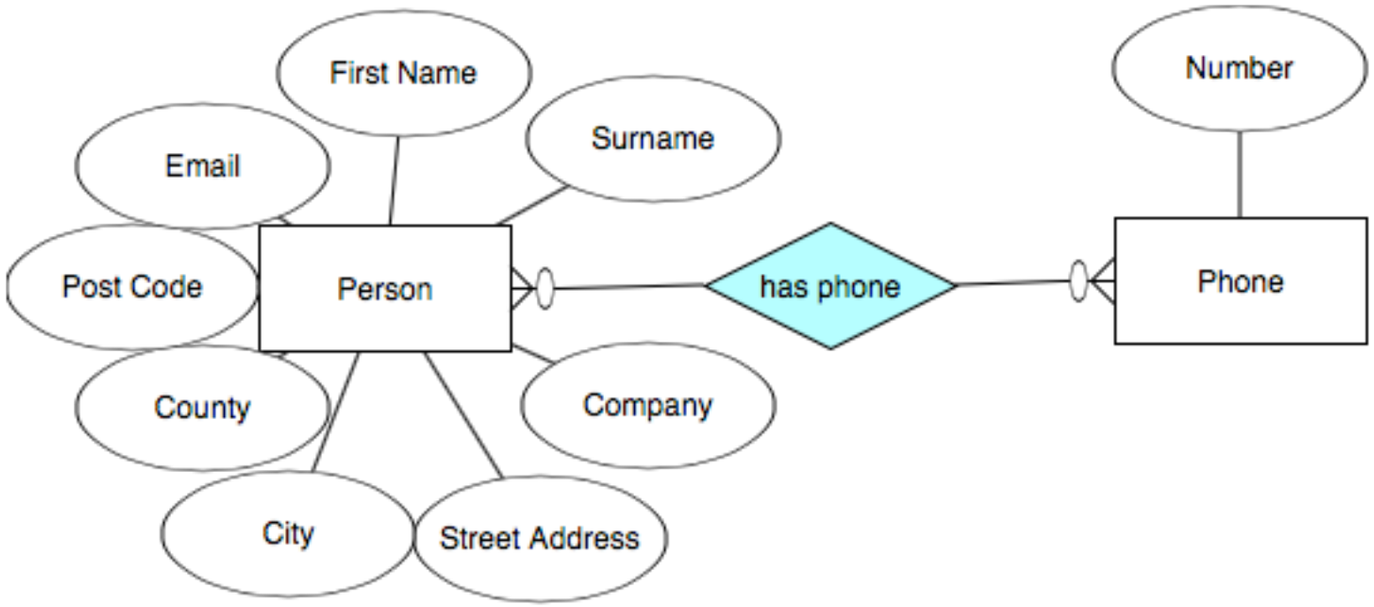
1st DM: Flat File

- **Domain**: People, addresses, personal data
  - in 1 (flat) csv file
- **Pain Points:**
  - variable numbers of the "same" attribute
    - phone number
    - email address
    - …
  - inserting columns is painful
  - partial columns/NULL values aren't great
  - companies have addresses
    - more than one!
    - and phone numbers, etc.



No data integrity guarantee!

From Flat File towards 2nd DM: Relational



- Better Format
  - two 2 (flat) csv files
- **Pain Points:**
  - sorting destroys the relationship
    - we used row numbers to connect the 2 files
    - sorting changes the row number!
  - hard to see the record
  - no longer a flat file
  - CSV format makes assumptions

2nd DML: Relational Model for Addresses

- M1
  1. Design a conceptual model for this domain
  2. normalise it
  3. create different tables for suitable aspects of this domain
  4. linked via “foreign keys” offered by relational formalism

➡ no more pain points:

- this domain fits nicely our “table” relational data model (RDM)
- RDM also comes with a suitable
  - **data manipulation language** for
    - querying
    - sorting
    - inserting tuples
  - **schema language**
    - constraining values
    - expressing functional/key constraints

SQL

And with
data integrity guarantee!

From Relational to XML (1)

- **Domain**: People, addresses, management structure
  - in relational/SQL tables
- **2 Pain points:**
  1. (cumbersome) querying - it requires (too) many joins!
  2. (nigh impossible) ensuring integrity - unbounded ‘manages’ paths require **recursive** queries/joins to avoid cyclic management structure

Complicated to write/ maintain queries

Employees

Employee ID	Postcode	City	…
1234123	M16 0P2	Manchester	…
1234124	M2 3OZ	Manchester	…
1234567	SW1 A	London	…
...	...	...	...

Management

Manager ID	ManageeID
1234124	1234123
1234567	1234124
1234123	1234567
...	...

From Relational to XML (2)

- **Domain**: Proteins
- **Pain points:**
  - cumbersome:
    - querying: too many joins!

Protein ID	Full Name	Short	Organism	...
1234123	Fanconi anemia group J	FACJ	Halorubrum phage	...
1234567	ATP-dependent	N/A	Gallus gallus / Chicken	...
	...	...	...	...

...

Protein ID	Alternative Name
1234123	ATP-dependent RNA helicase BRIP1
1234123	BRCA1-interacting protein C-terminal helicase 1
1234123	BRCA1-interacting protein 1
	...

Protein	Genes
1234123	BRIP1
1234123	BACH1
1234567	helicas
	...

Graph-based Data Models

New Domains

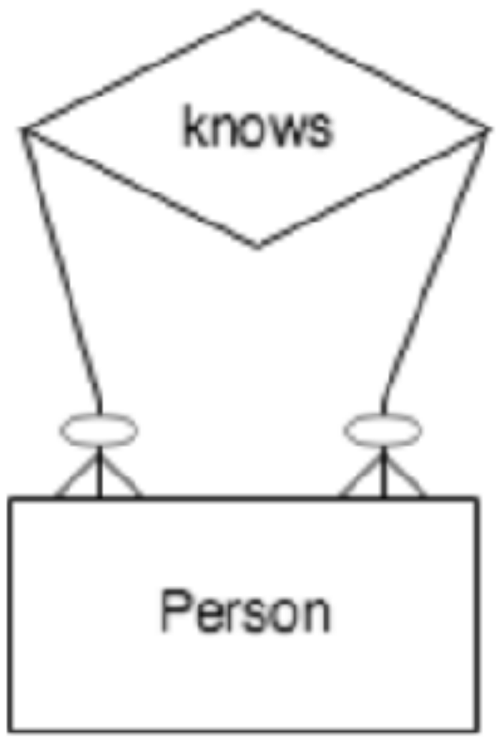
- with new requirements:
- Sociality
  - friend-of/knows/likes/acquainted-with/trusts/…
  - works-with/colleague-of/…
  - interacts-with/reacts-with/binds-to/activates/…
  - student-of/fan-of/…
  - cites
  - …
  - such relationships form social/professional/bio-chemical/adademic *networks*
  - we focus on **social** here: knows
- How are they different to “manages”
- How do we capture these?

Draw an ER diagram of social networks involving

- people
- knows

“Knows” in SQL - ER Diagram

simple:



“Knows” in SQL tables

```
CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

```
CREATE TABLE knows
(
Who int,
Whom int,
FOREIGN KEY (Who)
    REFERENCES Persons(P_Id),
FOREIGN KEY (Whom)
    REFERENCES Persons(P_Id)
);
```

not optimal - remember W1

“Knows” in SQL - Queries (1)

```
CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

```
CREATE TABLE knows
(
Who int,
Whom int,
FOREIGN KEY (Who)
      REFERENCES Persons(P_Id),
FOREIGN KEY (Whom)
      REFERENCES Persons(P_Id)
);
```

“friends of Bob Builder”

How many people does Bob Builder know?

```
SELECT COUNT(DISTINCT k.Whom)
FROM Persons P, knows k
WHERE ( P.PersonID = k.Who AND
            P.FirstName = “Bob” AND
            P.LastName = “Builder” );
```

“Knows” in SQL - Queries (2)

```
CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

```
CREATE TABLE knows
(
Who int,
Whom int,
FOREIGN KEY (Who)
      REFERENCES Persons(P_Id),
FOREIGN KEY (Whom)
      REFERENCES Persons(P_Id)
);
```

Give me the names of Bob Builder’s friends?

```
SELECT P2.FirstName , P2.LastName
FROM knows k, Persons P1, Persons P2
WHERE ( P1.FirstName = “Bob” AND
            P1.LastName = “Builder” AND
            P1.PersonID = k.Who AND
            P2.PersonID = k.Whom  );
```

“Knows” in SQL - Queries (3)

```
CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

```
CREATE TABLE knows
(
Who int,
Whom int,
FOREIGN KEY (Who)
      REFERENCES Persons(P_Id),
FOREIGN KEY (Whom)
      REFERENCES Persons(P_Id)
);
```

Give me the names of Bob Builder’s friends’ friends?

```
SELECT P3.FirstName , P3.LastName
FROM knows k1, knows k2, Persons P1, Persons P3
WHERE (  P1.FirstName = “Bob” AND
         P1.LastName = “Builder” AND
         k1.whom = k2.who AND
         P1.PersonID = k1.Who AND
         P3.PersonID = k2.Whom );
```

“Knows” in SQL - Queries (4)

```
CREATE TABLE Persons
(
PersonID int,
LastName varchar(255),
FirstName varchar(255),
Address varchar(255),
City varchar(255)
);
```

```
CREATE TABLE knows
(
Who int,
Whom int,
FOREIGN KEY (Who)
      REFERENCES Persons(P_Id),
FOREIGN KEY (Whom)
      REFERENCES Persons(P_Id)
);
```

aaargh remember Week2?

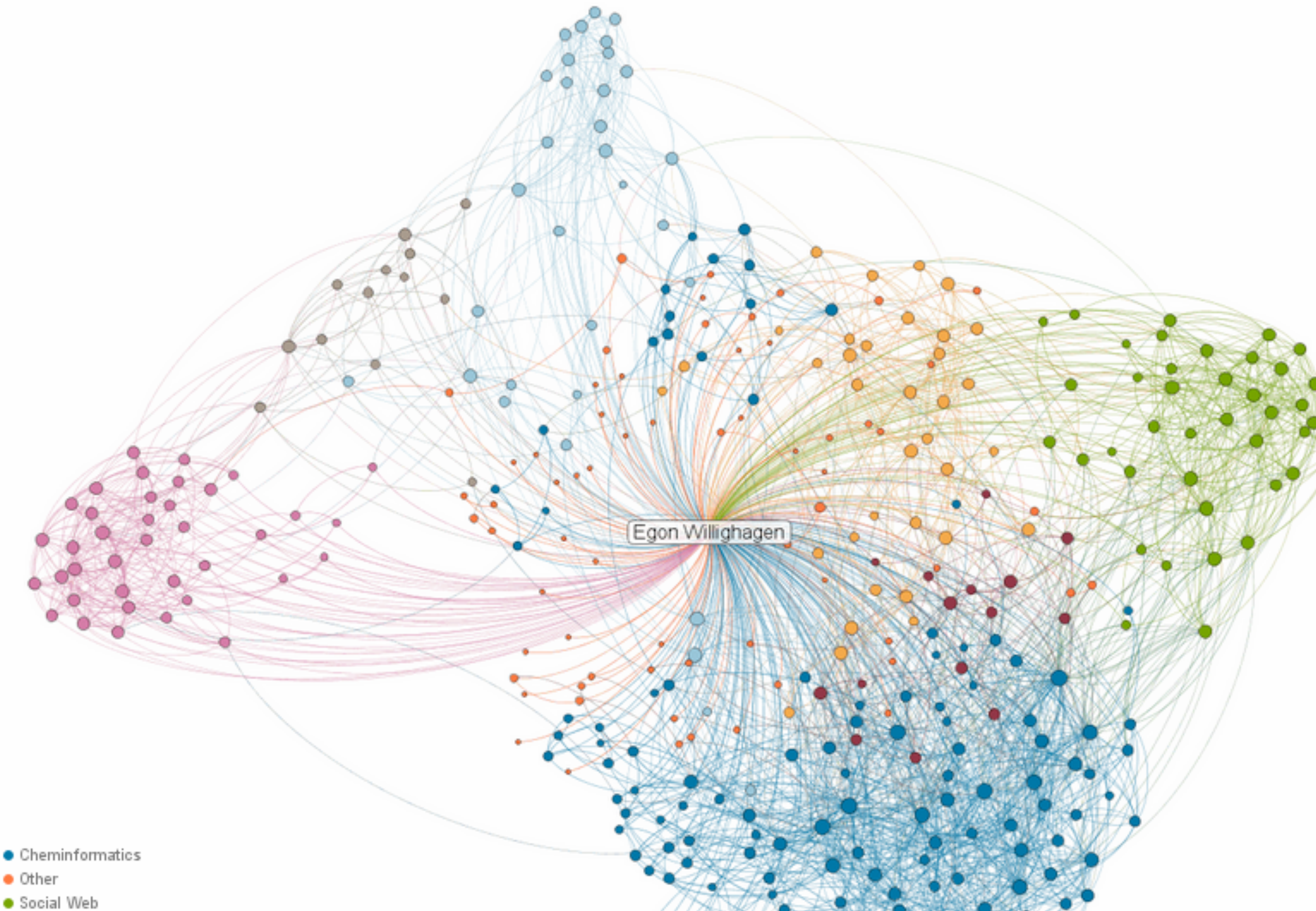
paths of *unbounded* length!

Give me the names of everybody in Bob Builder’s **network**?

```
SELECT P3.FirstName , P3.LastName
FROM knows k1, knows k2, knows k3,….Persons P1, Persons P3
WHERE (  (k1.whom = k2.who OR k1.whom = P3.PersonID) AND
         (k2.whom = k3.whom OR  k2.Whom = P3.PersonID) AND
            …..
         P1.FirstName = “Bob” AND
         P1.LastName = “Builder” );
```

LinkedIn Maps

Egon Willighagen's Professional Network
as of January 25, 2011



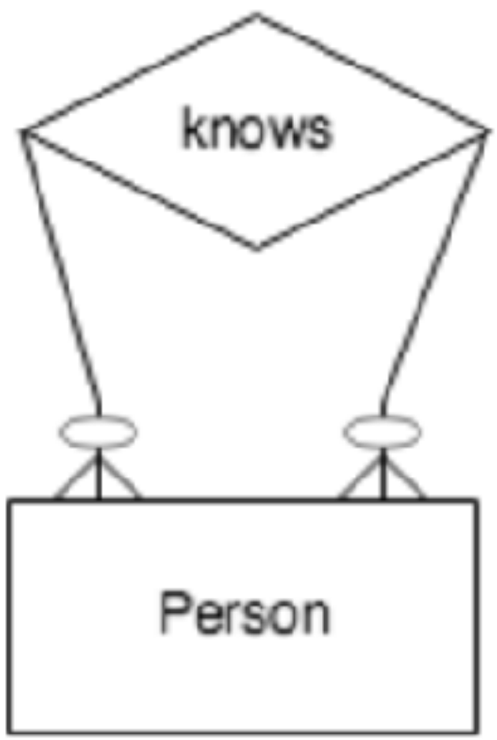
"Knows" in SQL - Pain Points

- **Fundamental** Pain Points:
  - variable number of "relationships" ⇨ split tables/normalise
    - ➡ queries require joins
    - ➡ performance may deteriorate & queries become error prone
  - domain may require *unbounded joins*
    - to explore a network of friends/paths of unbounded length
    - requires recursive queries or bounds on domain structure

- **Technology Specific** Pain Points:
  - does your SQL DBMS support
    - recursive queries?
    - transitive closure?
  - if yes: fine
  - if not: we can't query whole, unbounded networks!

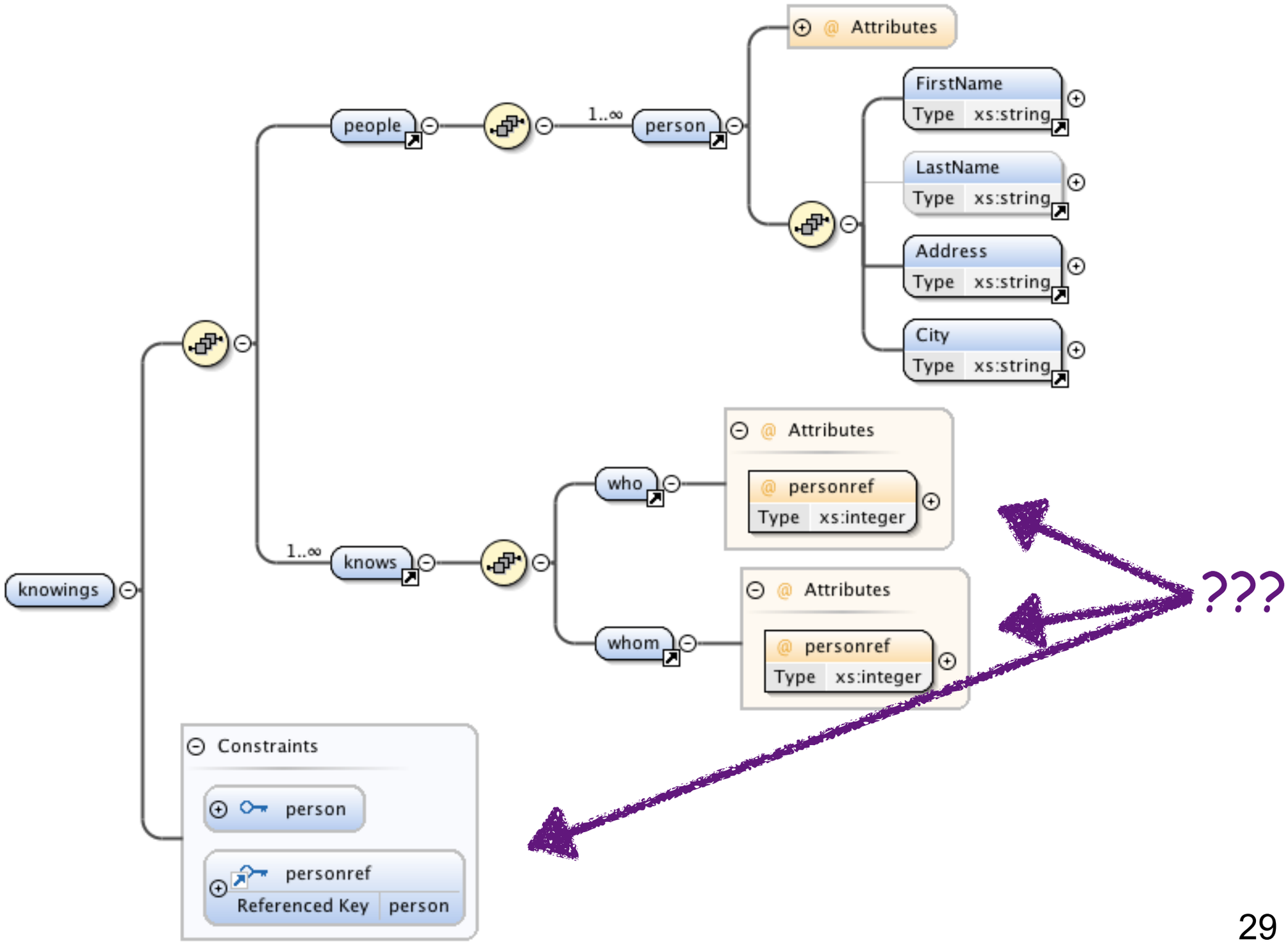
“Knows” in XML

- Of course we still have the same conceptual model



- And let’s follow the SQL for the logical model/schema!

Knowings WXS



Example Document & WXS

```
<knowings>
  <people>
    <person id="1">
      <FirstName>Bob</FirstName>
      <LastName>Builder</LastName>
      <Address>Some…</Address>
      <City>Manchester</City>
    </person>
    <person id="2">
      <FirstName>Wendy</FirstName>
      <Address>…rainbow</Address>
      <City>Manchester</City>
    </person>
  </people>
  <knows>
    <who personref="1"/>
    <whom personref="2"/>
  </knows>
</knowings>
```

```
<xs:element name="person">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="FirstName" type="xs:string"/>
        …
      </xs:sequence>
      <xs:attribute name="id" type="xs:ID" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="knows">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="who">
          <xs:complexType>
            <xs:attribute name="personref" type="xs:IDREF"
                          use="required"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="whom">
          <xs:complexType>
            <xs:attribute name="personref" type="xs:IDREF"
                          use="required"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Counting Friends!

How many friends does Bob Builder have?

```
SELECT COUNT(DISTINCT k.Whom)
FROM Persons P, knows k
WHERE ( P.PersonID = k.Who AND
        P.FirstName = “Bob” AND
        P.LastName = “Builder” );
```

```
count(
    //whom
        [../who/@personref =
            //person[FirstName="Bob"
                and LastName="Builder"]/@id])
```

Bob’s id

Get those friends!

Give me the names of Bob Builder's friends?

```
SELECT P2.FirstName , P2.LastName
FROM knows k, Persons P1, Persons P2
WHERE ( P1.PersonID = k.Who AND
        P2.PersonID = k.Whom AND
        P1.FirstName = "Bob" AND
        P1.LastName = "Builder" );
```

First: get the whole person (who's friend with BB)

```
//person[@id =
    //whom
        [../who/@personref =
        //person[FirstName="Bob"
        and LastName="Builder"]/@id]/@personref
]
```

Bob's friends

Get those friends!

Give me the names of Bob Builder's friends?

```
SELECT P2.FirstName , P2.LastName
FROM knows k, Persons P1, Persons P2
WHERE ( P1.PersonID = k.Who AND
        P2.PersonID = k.Whom AND
        P1.FirstName = "Bob" AND
        P1.LastName = "Builder" );
```

Second: use a bit of XQuery to get their names

```
for $p in //person[@id =
      //whom
            [../who/@personref =
            //person[FirstName="Bob"
            and LastName="Builder"]/@id]/@personref
 ]
return <name>{$p/FirstName} {$p/LastName}</name>
```

Get those friends!

Function it up a bit

```
declare function local:friendsOf($person) {
        for $p in
        $person/../person[@id = //whom
                [../who/@personref = $person/@id]/@personref]
        return $p
};

declare function local:fullNameOf($person) {
   <name>{$person/FirstName} {$person/LastName}</name>
};

for $f in local:friendsOf(//person[FirstName="Bob"
                        and LastName="Builder"])

return local:fullNameOf($f)
```

All friends of friends

Give me the names of friends of friends of Bob Builder!

```
SELECT P3.FirstName , P3.LastName
FROM knows k1, knows k2, Persons P1, Persons P3
WHERE (  k1.whom = k2.who AND
         P1.PersonID = k1.Who AND
         P3.PersonID = k2.Whom AND
         P1.FirstName = “Bob” AND
         P1.LastName = “Builder” );
```

See next slide!

All friends of friends in Network

```
declare function local:friendsOf($person) {
    for $p in
    $person/../person[@id = //whom
            [../who/@personref = $person/@id]/@personref]
    return $p
};

declare function local:friendsOfFriend($person) {
   for $p in local:friendsOf($person)
   return
      if (empty($p))
         then $p (: done :)
         else  (local:friendOf($p))
};

declare function local:fullNameOf($person) {
   <name>{$person/FirstName} {$person/LastName}</name>
};


for $f in local:friendsOfFriend(//person[FirstName="Bob"
                    and LastName="Builder"])

return local:fullNameOf($f)
```

get friends of friends

All friends in Network

Give me the names of people in Bob Builder's network?

```
SELECT P3.FirstName , P3.LastName
FROM knows k1, knows k2, knows k3,….Persons P1, Persons P3
WHERE (  (k1.whom = k2.who OR k1.whom = P3.PersonID) AND
         (k2.whom = k3.whom OR  k2.Whom = P3.PersonID) AND
           …..
         P1.FirstName = "Bob" AND
         P1.LastName = "Builder" );
```

See next slide!

All friends in Network

```
declare function local:friendsOf($person) {
    for $p in
    $person/../person[@id = //whom
            [../who/@personref = $person/@id]/@personref]
    return $p
};

declare function local:friendTreeOf($person) {
   for $p in local:friendsOf($person)
   return
      if (empty($p))
         then $p (: Base case of the recursion! :)
         else  ($p, local:friendTreeOf($p))
};

declare function local:fullNameOf($person) {
   <name>{$person/FirstName} {$person/LastName}</name>
};


for $f in local:friendTreeOf(//person[FirstName="Bob"
                    and LastName="Builder"])

return local:fullNameOf($f)
```

get friends of friends recursion! of friends of ...

All friends in Network - is this robust?

```
declare function local:friendsOf($person) {
      for $p in
      $person/../person[@id = //whom
              [../who/@personref = $person/@id]/@personref]
      return $p
};

declare function local:friendTreeOf($p
   for $p in local:friendsOf($p
   return
      if (empty($p))
         then $p (: Base c
         else  ($p, local:fri
};

declare function local:fullNameOf($person) {
   <name>{$person/FirstName} {$person/LastName}</name>
};


for $f in local:friendTreeOf(//person[FirstName="Bob"
                     and LastName="Builder"])

return local:fullNameOf($f)
```

No - does not terminate in case of cycles in network: infinite loop/stack overflow!

```
<knowings>
   <people>
      <person id="1">
         <FirstName>Bob</FirstName>

                  >
                  d="2">
                  me>Wendy</FirstName>

      <person id="3">
         <FirstName>Cindy</FirstName>
            …
      </person>
   </people>
   <knows>
      <who personref="1"/><whom personref="2"/>
   </knows>
   <knows>
      <who personref="2"/><whom personref="3"/>
   </knows>
   <knows>
      <who personref="3"/><whom personref="1"/>
   </knows>
</knowings>
```

Cycles Cause Problems

- We now have to implement **cycle detection**
  - into *local:friendTreeOf(…)*
  - and perhaps some other stuff!
- **New** pain points
  - Identity of node through 1 relation was tough
    - Managing the IDs, personrefs, etc. was...unpleasant
    - If we add other sorts of nodes, could get more *tedious*
      - ID, IDREF was tricky enoug
      - Key and Keyref are even touch challenging!
    - error prone!
  - Tree like sets were ok, but cycles are hard
    - This will be true for formats like “GraphML”!

Choices!

"Knowings"?
Really?

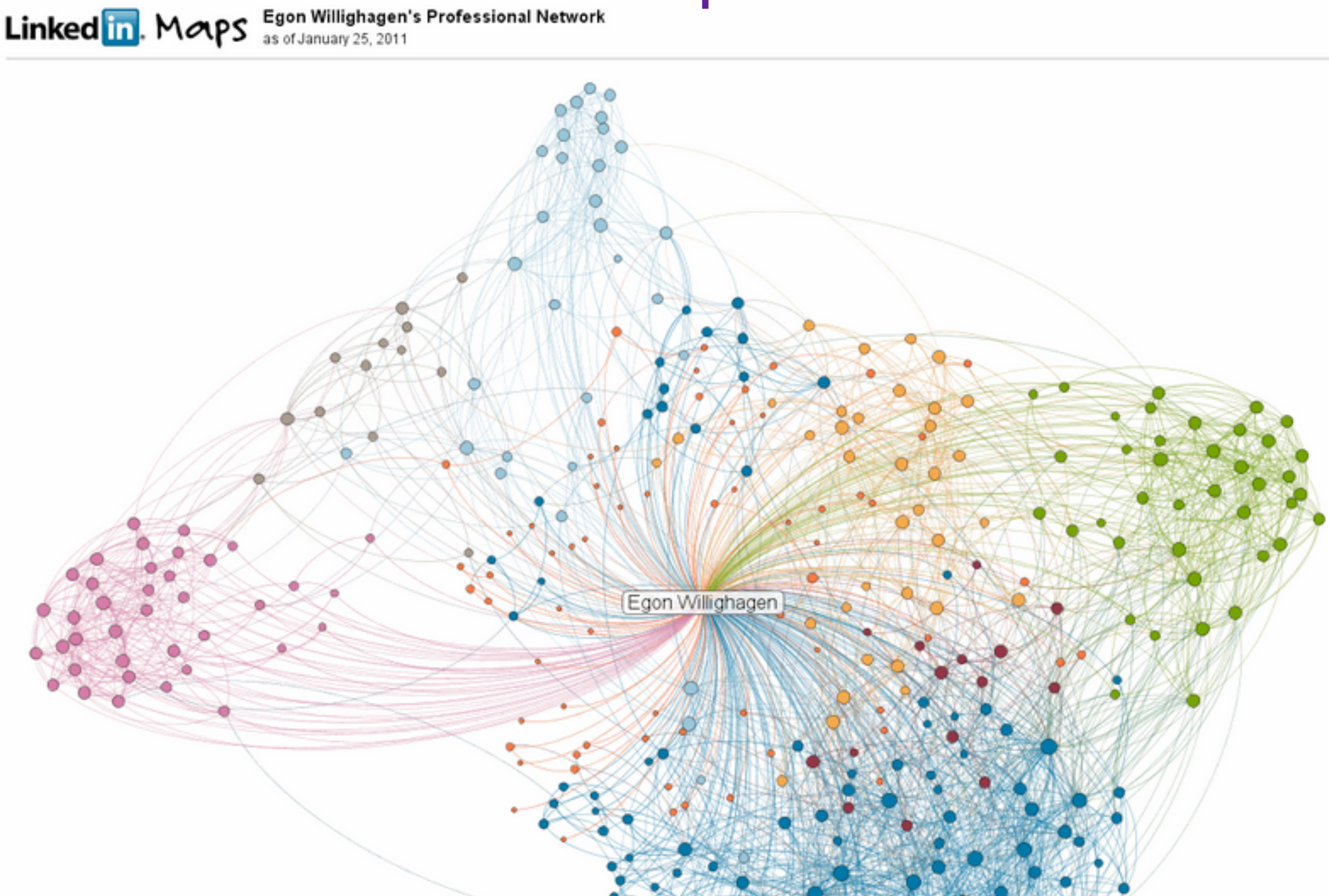
```
<knowings>
  <people>
    <person id="1">
      <FirstName>Bob</FirstName>
      <LastName>Builder</LastName>
      <Address>Somewhere Cool</Address>
      <City>Manchester</City>
    </person>
    <person id="2">
      <FirstName>Wendy</FirstName>
      <Address>88 Jackson Crescent</Address>
      <City>Manchester</City>
    </person>
  </people>
  <knows>
    <who personref="1"/>
    <whom personref="2"/>
  </knows>
</knowings>
```

Why People but "knows" as direct child?

None of these issues touch the data structure mismatch problem

Couldn't we just embed who each person knows in that element?

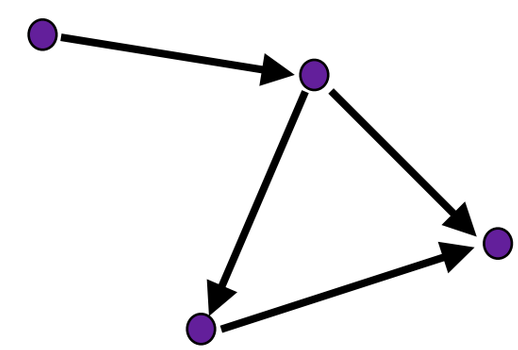
“Knows” forms a Graph



Graph Basics (1)

- A **graph** G = (V,E) is a pair with
  - V a set of **vertices** (also called) **nodes**, and
  - E ⊆ V × V a set of **edges**

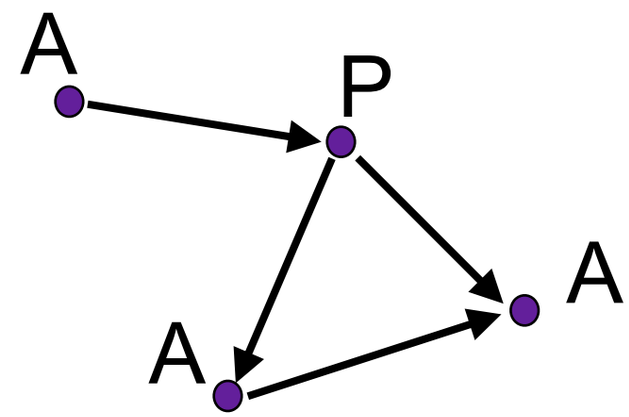
- Example: G = ({a,b,c,d}, {(a,b), (b,c), (b,d), (c,d)})
  - where are a,….d in this graph's picture?



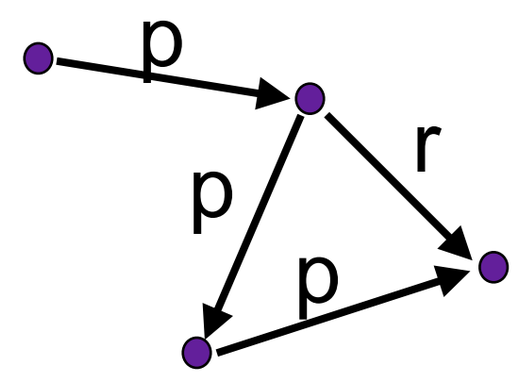
- Variants:
  - (in)finite graphs: V is a (in)finite set
  - (un)directed graphs: E (is) is not a symmetric relation
    - i.e., if G is undirected, then (x,y) ∈ E implies (y,x) ∈ E.
  - node/edge labelled graphs: a label set S, labelling function(s)
    - $\mathcal{L}$: V → S (node labels)
    - $\mathcal{L}$: E → S (edge labels)

Graph Basics (2)

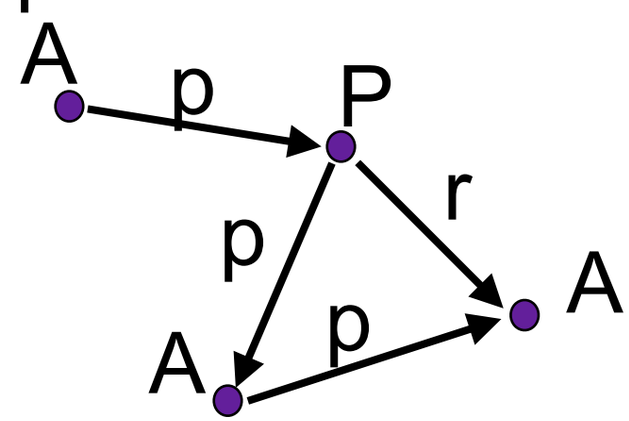
- Example: node-labelled graph
  - $\mathcal{L}$: V → {A,P}



- Example: edge-labelled graph
  - $\mathcal{L}$: E → {p,r,s}

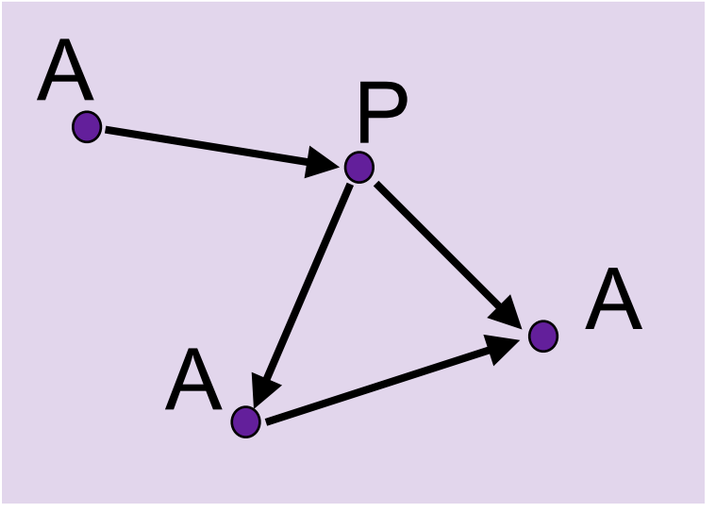


- Example: node-and-edge-labelled graph
  - $\mathcal{L}$: V → {A,P}
  - $\mathcal{L}$: E → {p,r,s}



Graph Basics (3)

- Pictures are a BAD external representation for graphs

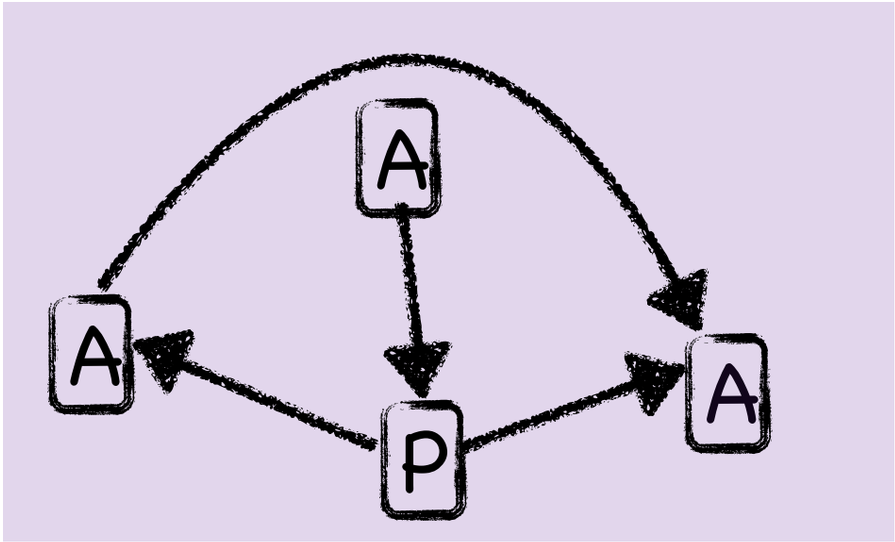


=

$$G = (\{a,b,c,d\},\ \{(a,b), (b,c), (b,d), (c,d)\},\ \mathcal{L}: V \rightarrow \{A,P\}\ \ \mathcal{L}: a \mapsto A,\ b \mapsto P,\ c \mapsto A,\ d \mapsto A\ )$$

=

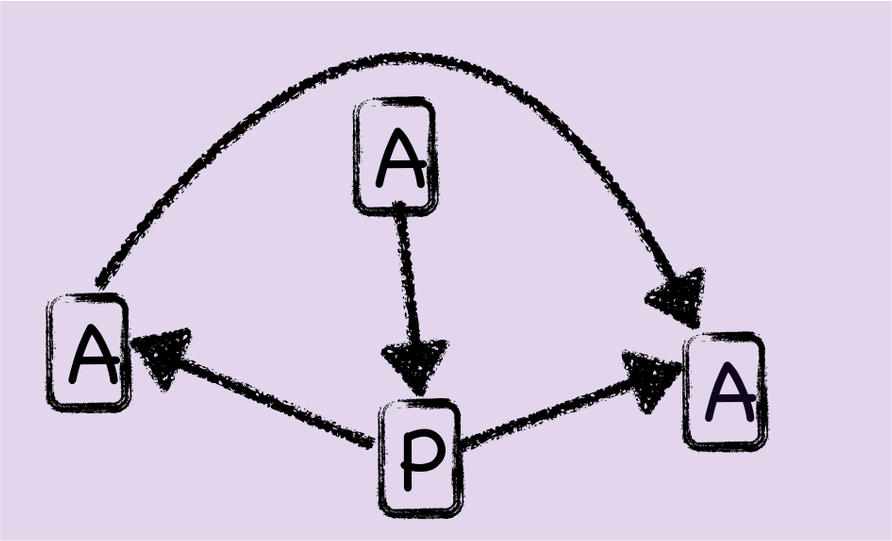
=



= …

Graph Basics (4)

- Pictures are a BAD **external representation** for graphs
  - it captures loads of irrelevant information
    - colour
    - location, geometry,
    - shapes, strokes, …
  - what if labels are more complex/structured?
  - how do we *parse* a picture into an **internal representation***?*

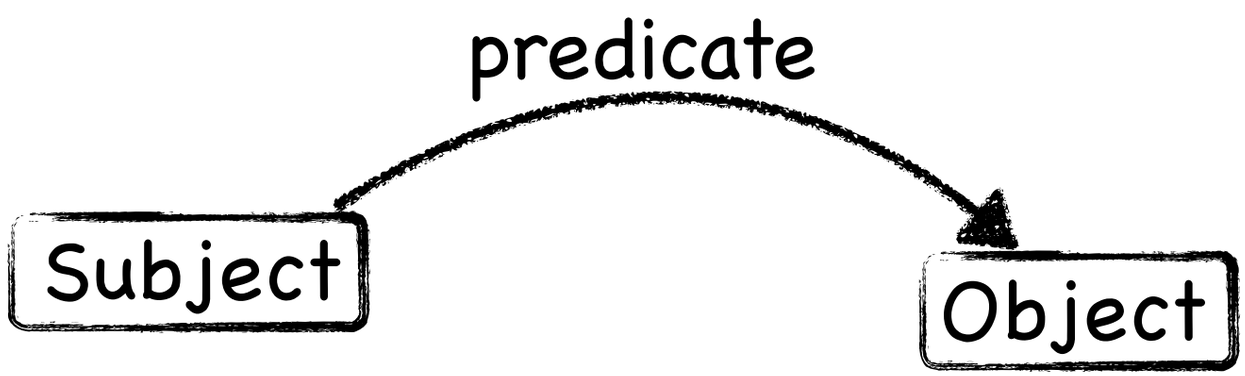


RDF
a data structure formalisms for graphs

A Graph Formalism: RDF

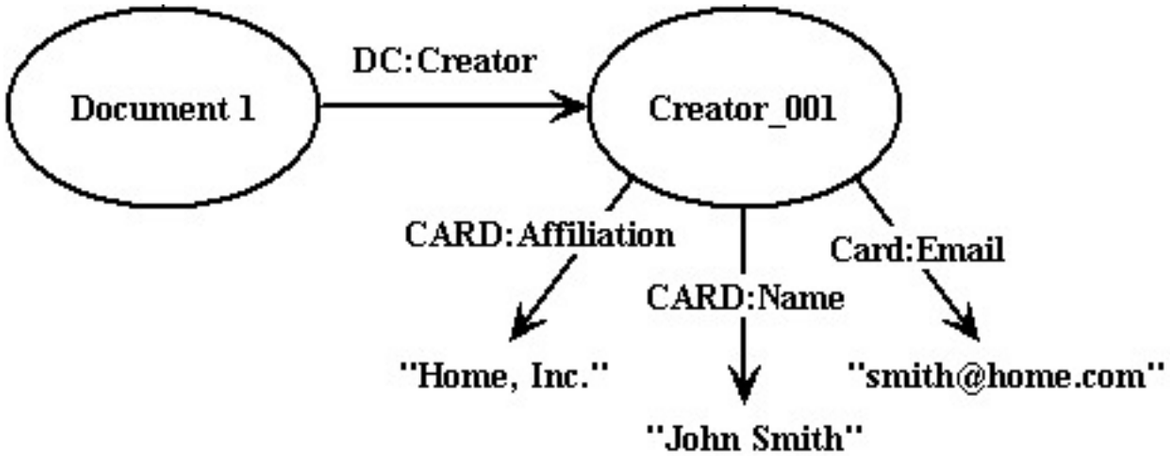
- **R**esource **D**escription **F**ramework
- a **graph-based** data structure formalism
- a W3C standard for the representation of **graphs**
- comes with various syntaxes for ExtRep
- is based on **triples**

(subject, predicate, object)



Resource Description

- RDF = **R**esource **D**escription **F**ramework
- A resource is
  "*any object that is uniquely identifiable by a Uniform Resource Identifier (URI)*"
  - e.g., a person, cat, book, article, protein, painting,…



http://www.dlib.org/dlib/may98/miller/05miller.html

RDF: basics

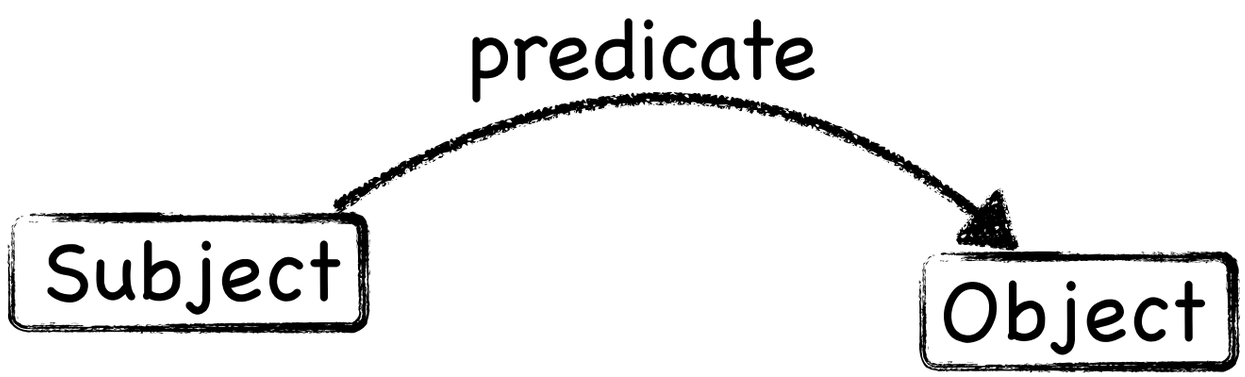
- an RDF **graph G** is a **set** of **triples**

$$\{(s_i, p_i, o_i) \mid 1 \leq i \leq n\}$$

- where each
  - $s_i \in U \cup B$
  - $p_i \in U$
  - $o_i \in U \cup B \cup L$

U: URIs (for resources), incl. rdf:type
B: Blank nodes
L: Literals (used for values such as strings, numbers, dates)

(subject, predicate, object)



RDF: an example

- an RDF **graph G** is a **set** of **triples**

$$\{(s_i, p_i, o_i) \mid 1 \leq i \leq n\}$$

- where each
  - $s_i \in U \cup B$, $p_i \in U$, $o_i \in U \cup B \cup L$

U: URIs (for resources)
B: Blank nodes
L: Literals

{(ex:bparsia, foaf:knows, ex:bparsia),
(ex:bparsia, rdf:type, foaf:Person),
(ex:bparsia, rdf:type, Agent),
(ex:sattler, foaf:title, “Dr.”),
(ex:bparsia, foaf:title, “Dr.”),
(ex:sattler, foaf:knows, ex:alvaro),
(ex:bparsia, foaf:knows, ex:alvaro) }

a graph ???

abbreviate: ex: for http://www.cs.man.ac.uk/
foaf: for http://xmlns.com/foaf/0.1/

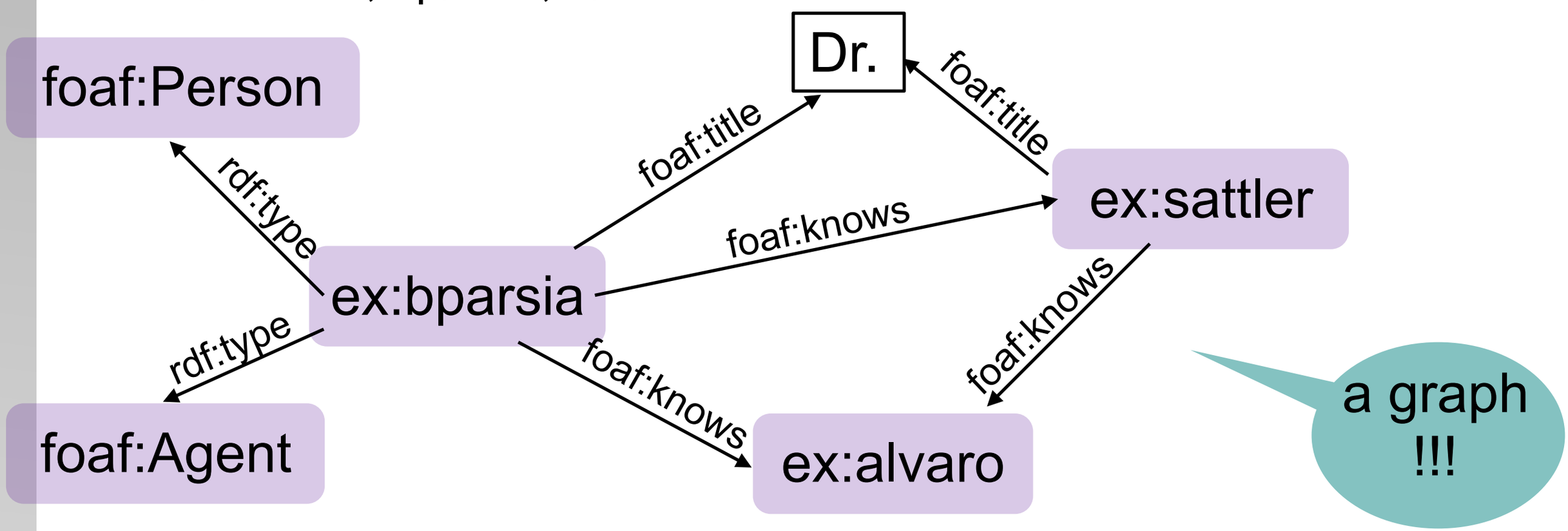
RDF: an example (2)

- an RDF **graph G** is a **set** of **triples**

$$\{(s_i, p_i, o_i) \mid 1 \leq i \leq n\}$$

- where each
  - $s_i \in U \cup B$, $p_i \in U$, $o_i \in U \cup B \cup L$

U: URIs (for resources)
B: Blank nodes
L: Literals



abbreviate: ex: for http://www.cs.man.ac.uk/
foaf: for http://xmlns.com/foaf/0.1/

RDF syntaxes

- "serialisation formats"
  - External Representations of RDF graphs
- there are various:
  - **Turtle**
  - N-Triples
  - JSON-LD
  - N3
  - RDF/XML
  - …
- plus translators between them!
- our example is **not** in any of these:

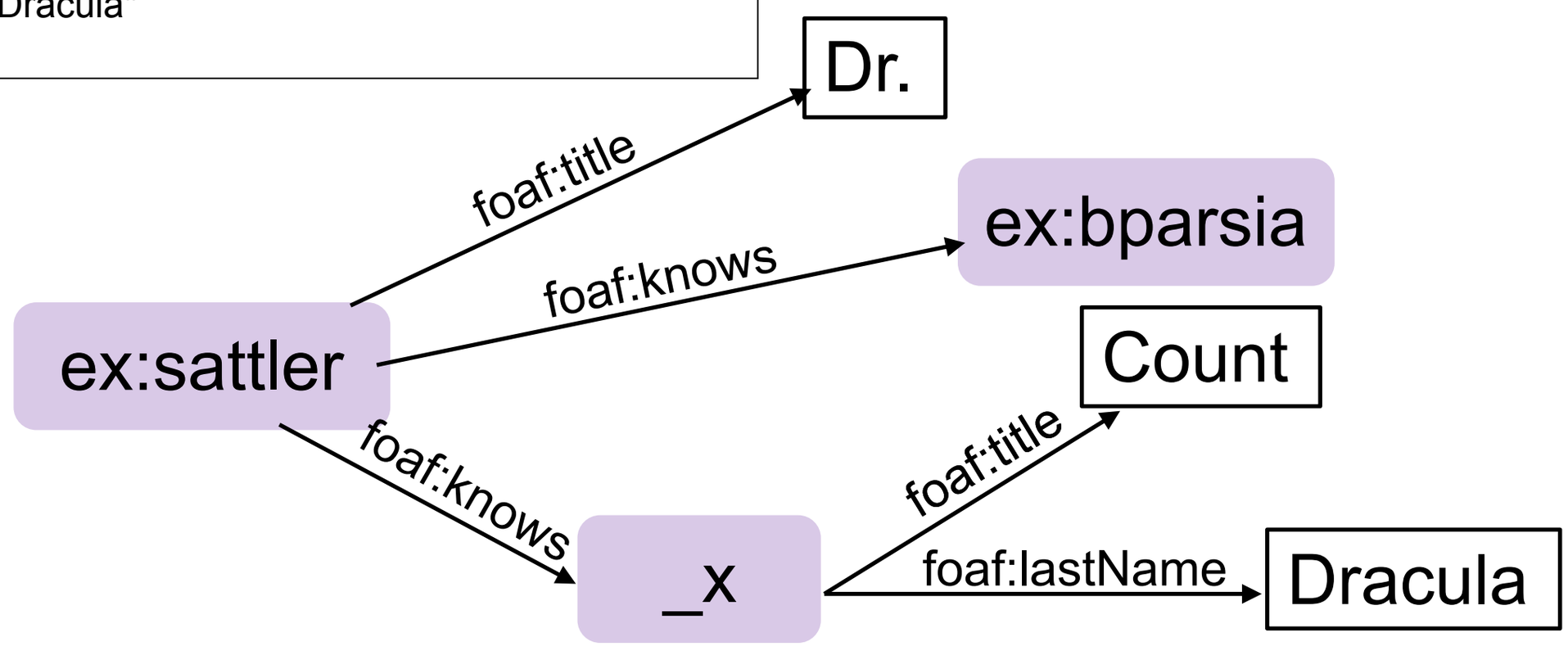
5 triples in Turtle:

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

ex:sattler
  foaf:title  "Dr." ;
  foaf:knows ex:bparsia ;
  foaf:knows
   [
    foaf:title "Count";
    foaf:lastName "Dracula"
  ] .
```

~~{(ex:bparsia, foaf:knows, ex:bparsia/),
(ex:bparsia, rdf:type, foaf:Person),
…}~~

RDF syntaxes - Turtle

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

ex:sattler
  foaf:title  "Dr." ;
  foaf:knows ex:bparsia ;
  foaf:knows
   [
    foaf:title "Count";
    foaf:lastName "Dracula"
  ] .
```

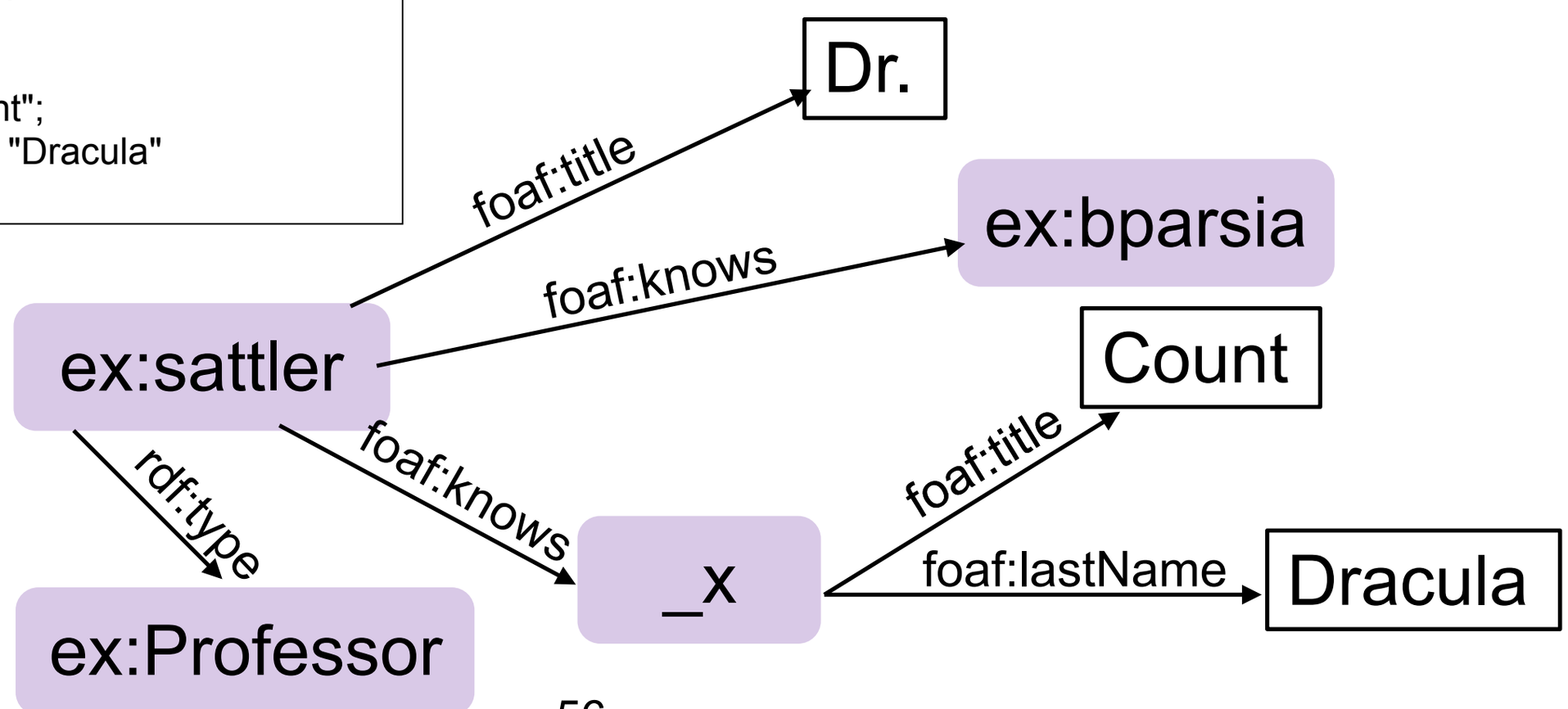


RDFS
a schema language for RDF

RDFS: A different sort of schema

- in RDF, we have rdf:type

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

ex:sattler
  rdf:type ex:Professor
  foaf:title  "Dr." ;
  foaf:knows ex:bparsia ;
  foaf:knows
   [
    foaf:title "Count";
    foaf:lastName "Dracula"
  ] .
```



RDFS: A different sort of schema

- in RDF, we have rdf:type
- RDF**S** is a **schema language** for RDF
- in RDFS, we also have
  - rdfs:subClassOf
    - e.g. (ex:Professor, rdfs:subClassOf, foaf:Person), (foaf:Person, rdfs:subClassOf, foaf:Agent)
  - rdfs:subPropertyOf
    - e.g. (ex:hasDaughter, rdfs:subPropertyOf, ex:hasChild)
  - rdfs:domain
    - e.g. (ex:hasChild, rdfs:domain, foaf:Person)
  - rdfs:range
    - e.g. (ex:hasChild, rdfs:range, foaf:Person)

no 's'

's'

Inference: Default Values++

- RDFS does **not** *describe/constrain structure*
  - unlike XML style schema languages, RDFS can't be used to "validate" documents/graphs
    - at least easily
    - primary goal of RDFS is *adding extra information*
    - … like default values (but different)!

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

ex:sattler
  foaf:title  "Dr." ;
  foaf:knows ex:bparsia ;
  foaf:knows
   [
    foaf:title "Count";
    foaf:lastName "Dracula"
  ] .
```

+

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:Person rdfs:subClassOf foaf:Agent
```

=>

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

ex:sattler rdf:type foaf:Person.
ex:sattler rdf:type foaf:Agent
ex:bparsia rdf:type foaf:Person.
ex:bparsia rdf:type foaf:Agent
```

Inference: Default Values++

- RDFS does **not** *describe/constrain structure*
  - That is, unlike XML style schema languages, RDFS can't be used to "validate" documents/graphs
    - at least easily
    - The primary goal of RDFS is *adding extra information*
    - … like default values (but different)!

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

ex:sattler
  rdf:type ex:Professor
  foaf:title  "Dr." ;
  foaf:knows ex:bparsia ;
  foaf:knows
   [
    foaf:title "Count";
    foaf:lastName "Dracula"
  ] .
```

+

```
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix ex:   <http://www.cs.man.ac.uk/> .

ex:Professor rdfs:subClassOf foaf:Person
foaf:knows rdfs:domain foaf:Person.
foaf:knows rdfs:range foaf:Person.
foaf:Person rdfs:subClassOf foaf:Agent
```

=>

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

ex:sattler rdf:type foaf:Person.
ex:sattler rdf:type foaf:Agent
ex:bparsia rdf:type foaf:Person.
ex:bparsia rdf:type foaf:Agent
```

For more on inference...

- ...we invite you to take our courses from the Ontology Engineering and Automated Reasoning theme:
  - COMP62342 Ontology Engineering for the Semantic Web
  - COMP60332 Automated Reasoning and Verification

SPARQL
a query language for graphs

SPARQL

- We have
  - A data structure (RDF)
    - graph-based one
  - A data definition language
    - not really but sort of: RDFS
    - plus loads of external representations
  - Part of manipulation
    - Insert/authoring (RDF)
    - We need query!
- SPARQL
  - Standardised query language for RDF
    - Not the only graph query language out there!
    - E.g., neo4j has it's own language "Cypher"
      - http://neo4j.com/developer/cypher/
      - Has "graph structural" features like "shortest path"

SPARQL: Basic Graph Patterns

- SPARQL is based on **graph patterns**
- A set of Turtle statements is a **basic graph pattern** (BGP)
  - e.g. {ex:sattler rdf:type foaf:Person}
  - (We put it in braces here!)
- in a BGP, we can replace URIs, bNodes, or Literals with *variables,* and this yields another BGP
  - e.g., {?x rdf:type foaf:Person}
  - e.g., {?x foaf:knows ?y. ?y foaf:knows ?z. ?z foaf:knows ?x}

SPARQL: Clauses (1)

- We combine a BGP with a query type
  - ASK
    - E.g., ASK WHERE {ex:sattler rdf:type foaf:Person}
    - Returns true or false (only)
  - SELECT
    - E.g., SELECT ?p WHERE {?p rdf:type foaf:Person}
    - Very much like SQL SELECT
- Note:
    - ASK returns a Boolean value (not an RDF graph!)
    - SELECT returns a table (not an RDF graph!)
    - SPARQL is *not* closed over graphs!
      - Very weird!

SPARQL Clauses (2)

- There are two query types that return graphs
  - CONSTRUCT
    - E.g., CONSTRUCT {?p rdf:type :Befriended}
      WHERE {?p foaf:knows ?q}
    - Like XQuery element and attribute constructors
  - DESCRIBE
    - E.g., DESCRIBE ?p
      WHERE {?p rdf:type foaf:Person}
    - Implementation dependent!
    - A “description” (as a graph)
      - Whatever the service deems helpful!
      - A bit akin to querying system tables in SQL

Example Data

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

ex:bobthebuilder
        foaf:firstName "Bob";
        foaf:lastName "Builder";
        foaf:knows ex:wendy ;
        foaf:knows ex:farmerpickles;
        foaf:knows ex:bijanparsia.
ex:wendy
        foaf:firstName "wendy";
        foaf:knows ex:farmerpickles.

ex:farmerpickles
        foaf:firstName "Farmer";
        foaf:lastName "Pickles";
        foaf:knows ex:bobthebuilder.

ex:bijanparsia
        foaf:firstName "Bijan";
        foaf:lastName "Parsia".
```

Counting Friends!

How many friends does Bob Builder have?

```
SELECT COUNT(DISTINCT k.Whom)
FROM Persons P, knows k
WHERE ( P.PersonID = k.Who AND
        P.FirstName = “Bob” AND
        P.LastName = “Builder” );
```

```
SELECT DISTINCT COUNT(?friend)
WHERE {?z
        foaf:firstName "Bob";
        foaf:lastName "Builder";
        foaf:knows ?friend };
```

Friends network?

Give me Bob Builder's friends' friends?

```
SELECT P3.FirstName , P3.LastName
FROM knows k1, knows k2, Persons P1, Persons P3
WHERE (  k1.whom = k2.who AND
         P1.PersonID = k1.Who AND
         P3.PersonID = k2.Whom AND
         P1.FirstName = "Bob" AND
         P1.LastName = "Builder" );
```

```
SELECT ?first, ?last
WHERE {?bobthebuilder
            foaf:firstName "Bob";
           foaf:lastName "Builder";
           foaf:knows ?middlefriend.
      ?middlefriend
            foaf:knows ?friend.
      ?friend foaf:firstName ?first;
              foaf:lastName ?last}
```

Friends network?

Give me Bob Builder's network?

```
SELECT P3.FirstName , P3.LastName
FROM knows k1, knows k2, Persons P1, Per
WHERE ( P1.FirstName = "Bob" AND
        P1.LastName = "Bui
    aaaaarrrrgh );
```

Sweet spot:
navigation via paths of
unbounded length
without cycle detection!

```
SELECT ?first,
WHERE {?bobti
        foaf:firstName "Bob";
        foaf:lastName "Builder";
        foaf:knows+ ?friend.
    ?friend foaf:firstName ?first;
            foaf:lastName ?last}
```

transitive closure

SPARQL and Inference

- SPARQL queries are sensitive to RDFS inference
  - as XPath is sensitive to default values!
  - also sensitive to more expressive language's inferences
    - like OWL!
      - in OWL, we can say that foaf:knows is *transitive*
      - so we don't necessarily need the property path to make our queries!
- Inference has a cost
  - results may be surprising
  - query answering may be computationally expensive!

Solves all problems?

- No!
  - We have to filter out Bob
    - to prevent getting him explicitly as his friend
    - because he may be in the cyclic paths
    - Foo!
      - But pretty easy with a FILTER
  - But pretty reasonable
    - **Path expressions** help a lot!
- Fairly normalised
  - sets of triples!
  - we don't get nice pre-assembled chunks like with XML
- No validation!
  - this is a formalism specific quirk
  - work is being done

Polyglot (Persistence)

- How can a format vary? How can **we** vary our format?
  - Same data model, same formalism, same implementation
    - But different formats, e.g., 2 XML-based address formats
  - Same data model, same formalism, same format
    - But different implementations, e.g., SQLite vs. MySQL
  - Same data model, same format
    - But different formalisms!
      - Usually, but not always, implies different implementations
      - XML in RDBMS
- We can be **explicitly** or **implicitly** polyglot
  - If we **encode** another data model into our “home” model
    - e.g., storing tables in XML <row>, <attribute>,…
    - We are still polyglot, but only implicitly so
    - Key Cost: Ad hoc implementation
  - If we split our domain model across multiple formalisms/implementations
    - We are **explicitly** poly
    - Key Cost: Model and System integration

Key points

- Understand your **domain**
  - What are you trying to represent and manipulate
- Understand the **fit** between domain and data model(s)
  - To see where there are sufficiently good fits
- Understand your infrastructure
  - And the cost of extending
- Understand integration vs. workaround costs

- Then make a *reasonable* decision
  - There will *always* be tradeoffs

Coursework for Week 5

- Due on Monday, November 5th, 9am

- Quiz 5 - the usual
- M 5: design a format
  - totally free
  - use SQL, JSON, XML, RDF, csv, Neo4J, …
  - for sharing ‘publishing’ information (articles, authors, etc)
  - make sure you understand the requirements
- CW 5: write some SPARQL queries
  - using Wikidata’s SPARQL endpoint
  - you will need to find codes (labels) for relations and terms
  - e.g., ‘is student of’ or ‘Painter’

Retrospective

Work in groups
on
4 Questions

Question 1: 30 mins

Which core data-model related **concepts/terms** and **properties** did you learn about?
And how are these related?

E.g. concepts: table, attribute, key, XML document, element, element name, attribute, schema, schema language, tree, PSVI, path, …

E.g. properties: robust (as such and in the face of change), extensible, faulty - in many different ways, scalable, round-trippable, well-formed, valid, self-describing, expressive, verbose, ...

Question 2: 30 mins

- Pick an application that requires some data sharing
  - e.g., cartoon sharing web site
- Design the architecture of your system
  - which main components are involved?
  - how does data flow/get checked/get stored?
  - how do you make this robust?
    - what kind of change do you plan for?
  - how polyglot is your system?

Question 3: 5 mins

Reflection:

Have you acquired new learning styles or skills?

Can you describe them?

Good Bye!

- We hope you have learned a lot!
- It was a pleasure to work with you!
- Speak to us about projects
  - taster
  - MSc
- Enjoy the rest of your programme
  - COMP62421 query processing
  - COMP62342 inference - semantic web
- See you in labs
  - for Week 5 exercises