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)
Formats for ExtRep of data (SE4)

- Consider 2 formats $F_1 = \langle DS_1, CM_1, S_1, D_1 \rangle$
  $F_2 = \langle DS_2, CM_2, S_2, D_2 \rangle$

- 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

• Recap of
  – data models
  – pain points
  – formats
  – schemas,…

• Graph-based Data Model:
  – RDF
  – RDFS, a schema language for RDF
    • but quite different from all other schema languages
  – SPARQL, a data manipulation mechanism for RDF

• Retrospective session
Graph shaped 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 we have discussed

• 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
• House occupancies
From Flat File to Relational (1)

- **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.
From Flat File to Relational (2)

• 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
Use Relational Model for this Domain

• M1

• Design a conceptual model for this domain
  – normalise it
  – create different tables for suitable aspects of this domain
  – 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
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

<table>
<thead>
<tr>
<th>Employees</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Employee ID</td>
<td>Postcode</td>
<td>City</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1234123</td>
<td>M16 0P2</td>
<td>Manchester</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1234124</td>
<td>M2 3OZ</td>
<td>Manchester</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1234567</td>
<td>SW1 A</td>
<td>London</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Management</th>
<th>Manager ID</th>
<th>ManageeID</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1234124</td>
<td>1234123</td>
</tr>
<tr>
<td></td>
<td>1234567</td>
<td>1234124</td>
</tr>
<tr>
<td></td>
<td>1234123</td>
<td>1234567</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
From Relational to XML (2)

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

<table>
<thead>
<tr>
<th>Protein ID</th>
<th>Full Name</th>
<th>Short Name</th>
<th>Organism</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234123</td>
<td>Fanconi anemia group J</td>
<td>FAC J</td>
<td>Halorubrum phage</td>
<td>...</td>
</tr>
<tr>
<td>1234567</td>
<td>ATP-dependent</td>
<td>N/A</td>
<td>Gallus gallus / Chicken</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein ID</th>
<th>Alternative Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234123</td>
<td>ATP-dependent RNA helicase</td>
</tr>
<tr>
<td>1234123</td>
<td>BRCA1-interacting protein C-terminal helicase 1</td>
</tr>
<tr>
<td>1234123</td>
<td>BRCA1-interacting protein 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Protein ID</th>
<th>Genes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234123</td>
<td>BRIP1</td>
</tr>
<tr>
<td>1234123</td>
<td>BACH1</td>
</tr>
<tr>
<td>1234567</td>
<td>helicas</td>
</tr>
</tbody>
</table>

...
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/…
    – …
    – such relationships form
      social/professional/bio-chemical/academic networks
    – we focus on social here: knows

• How are they different to “manages”

• How do we capture these?
“Knows” in SQL - ER Diagram

simple:

[Diagram of a simple ER diagram showing a single entity type "Person" and a relationship named "knows"]]
“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)
);
“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)
);

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

How many friends does Bob Builder have?
“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)
);

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 AND
);
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)
);

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 );

Give me the names of Bob Builder’s friends’ friends?
“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)
);

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” );

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

aaargh
remember Week2?
paths of unbounded length!
“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
<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>
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" );
```
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

```xquery
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!

```sql
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

```plaintext
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)
All friends in Network - is this robust?

```xml
<knowings>
  <people>
    <person id="1">
      <FirstName>Bob</FirstName>
      ...
    </person>
    <person id="2">
      <FirstName>Wendy</FirstName>
      ...
    </person>
    <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>
```

```
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)
```
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 enough
      – 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!

Why People but “knows” as direct child?

“Knowings”? Really?

None of these issues touch the data structure mismatch problem

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

```
<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>
```
“Knows” forms a Graph
Graph Basics

• A **graph** $G = (V,E)$ is a pair with
  – $V$ a set of **vertices** (also called) **nodes**, and
  – $E \subseteq V \times V$ a set of **edges**

• Example: $G = (\{a,b,c,d\}, \{(a,b), (b,c), (b,d), (c,d)\})$
  – where are $a,\ldots,d$ 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 $(x,y) \in E$ implies $(y,x) \in E$.
  – node/edge labelled graphs: a label set $S$, labelling function(s)
    • $\mathcal{L}: V \rightarrow S$ (node labels)
    • $\mathcal{L}: E \rightarrow S$ (edge labels)
Graph Basics (2)

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

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

- Example: node-and-edge-labelled graph
  - $\mathcal{L}: V \rightarrow \{A,P\}$
  - $\mathcal{L}: E \rightarrow \{p,r,s\}$
Graph Basics (3)

- Pictures are a BAD external representation for graphs

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

= ...

= ..
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

- **Resource Description Framework**
- 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 = **Resource Description Framework**
- A resource is "any object that is uniquely identifiable by an Uniform Resource Identifier (URI)"
  - e.g., a person, cat, book, article, protein, painting,…

[Diagram of RDF relationships]

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

- an RDF graph $G$ is a set of triples
  \[ \{(s_i, p_i, o_i) | 1 \leq i \leq n\} \]

- 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)

```
    Subject
       ^
     / \
    /   \
   p   \
   /    \
  /     \
  O     Object
```
RDF: an example

- an RDF **graph** \( G \) is a **set** of **triples**
  \[ \{(s_i, p_i, o_i) | 1 \leq i \leq n\} \]
- where each
  - \( s_i \in U \cup B, \quad p_i \in U, \quad o_i \in U \cup B \cup L \)

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

**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
  - \(s_i \in U \cup B\), \(p_i \in U\), \(o_i \in U \cup B \cup L\)

**Diagram:**

- `ex:bparsia` rdf:type foaf:Agent
- `ex:sattler` foaf:title Dr.
- `ex:alvaro` foaf:title a graph

**Abbreviations:**
- 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:

```
{(ex:bparsia, foaf:knows, ex:bparsia/),
 (ex:bparsia, rdf:type, foaf:Person),
 ...}
```

5 triples in Turtle:

```reason
@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"
    ] .
```
RDF syntaxes - Turtle

```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"
       ] .
```

Diagram:
- ex:sattler
  - foaf:title: Dr.
  - foaf:knows: ex:bparsia
  - foaf:knows: [foaf:title: Count; foaf:lastName: Dracula]
- ex:bparsia
  - foaf:knows: _X
- _X
  - foaf:knows: ex:sattler
  - foaf:title: Count
  - foaf:lastName: Dracula
RDFS
a schema language for RDF
RDFS: A different sort of schema

- In RDF, we have `rdf:type`

```rdfs
@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"
        ] .
```

Graph:
- `ex:sattler` connected to `ex:Professor`
- `ex:sattler` connected to `ex:bparsia`
- `ex:bparsia` connected to `Count`
- `Count` connected to `Dracula`
- `Count` connected to `x` via `foaf:knows`
- `x` connected to `Dracula` via `foaf:lastName`
RDFS: A different sort of schema

• in RDF, we have rdf:type
• **RDFS** 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)
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)!

```turtle
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@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:sattler
  foaf:title "Dr." ;
  foaf:knows ex:bparsia ;
  foaf:knows
    [ foaf:title "Count";
    foaf:lastName "Dracula"
    ] .
```

+ 

```turtle
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@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
```

=>

```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 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 inference…

• ...we cordially invite you to take course 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
    • 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
- Any set of Turtle statements is a basic graph pattern
  - 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
  - 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 (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 {ex:bobthebuilder 
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?

```sql
SELECT P3.FirstName , P3.LastName
FROM knows k1, knows k2, Persons P1, Persons P3
WHERE ( P1.FirstName = "Bob" AND
        P1.LastName = "Builder"
      
```

aaaaaarrrggh 
```
);```

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

• SPARQL queries are sensitive to RDFS inference
  – The way 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
  – May be surprising
  – 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
Poly-

• How can we vary?
  – Same data model, same formalism, same implementation
    • But different domain models!
  – Same data model, same formalism, same domain model
    • Different implementations, e.g., SQLite vs. MySQL
  – Same data model, same domain model
    • Different formalisms!
      – Usually, but not always, implies different implementations
      – XML in RDBMS

• We can be explicitly or implicitly poly-
  – If we encode another data model into our home model
    • We are still poly-
    • 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 point

- 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
Retrospective

Work in groups on 4 Questions
Question 1 - 20 mins

Which core data-model related concepts did you learn about - and how are these related?

E.g., table, attribute, key, XML document, element, element name, attribute, schema, schema language, tree, PSVI, path, …
Question 2 - 20 mins

We discussed numerous properties that a system, an XML document, a format, a schema language,... can have:

• list them
• what do they relate/apply to?
• how do they relate to each other?

Some example 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, ...
Think of an example information system that consumes and/or generates data (e.g., in RDF or XML):

can you draw an architecture diagram of one of those?
Question 4 - 20 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