Techniques for Large-scale Data

2019-2020, GK-4

Ontologies

OWL 2 Web Ontology Language

- a language for expressing ontologies

OWL 2 is not:

- a programming language
- a schema language for syntax conformance
- a database framework

OWL 2 Web Ontology Language Primer (Second Edition)

- https://www.w3.org/TR/2012/REC-owl2-primer-20121211/

What is an ontology?

- “a set of precise descriptive statements about some part of the world”
- What do we mean here by “world”?
  - “the domain of interest”
  - “the subject matter of the ontology”

- A vocabulary is a set of terms with fixed meaning
- A terminology provides a provides a vocabulary and states how terms are related

Ontology vs. Schema

- An **ontology** describes all terms in the domain of interest
- A **database schema** covers all concepts that are to be modelled
- (We wouldn't model synonyms like "person" and "human" in a database schema, but we might want to describe these terms and the relationship between them in an ontology. Declaring terms in different ontologies to be equivalent is useful in integration.)

Basic notions in OWL 2

Axioms the basic statements that an OWL ontology expresses

Entities elements used to refer to real world objects

Expressions combinations of entities to form complex descriptions from basic ones

Classes and instances

- Mary is an instance of the class of all persons

Functional-Style Syntax

```
ClassAssertion( :Person :Mary )
```

RDF/XML Syntax

```
<Person rdf:about="Mary"/>
```

Turtle Syntax

```
:Mary rdf:type :Person .
```

OWL/XML Syntax

```
<ClassAssertion>
   <Class IRI="Person"/>
   <NamedIndividual IRI="Mary"/>
</ClassAssertion>
```

Classes and instances

- Mary is an instance of the class of all persons

Functional-Style Syntax

```
ClassAssertion( :Person :Mary )
```

RDF/XML Syntax

```
<Person rdf:about="Mary"/>
```

Turtle Syntax

```
:Mary rdf:type :Person .
```

OWL/XML Syntax

```
<ClassAssertion>
   <Class IRI="Person"/>
   <NamedIndividual IRI="Mary"/>
</ClassAssertion>
```

IRI – Internationalized Resource Identifier

- A string of characters that unambiguously identifies a resource
- Hierarchical naming scheme ensures extensibility
- Like URIs, but allow a larger character set to be used in the string
- Web addresses (Uniform Resource Locators, or URLs) are a special of URI, and thus are a special type of IRI.

Class membership is not exclusive

- Mary can simultaneously be an instance of the class Person and also an instance of the class Woman

Functional-Style Syntax

```
ClassAssertion( :Woman :Mary )
```

RDF/XML Syntax

```
<Woman rdf:about="Mary"/>
```

Turtle Syntax

```
:Mary rdf:type :Woman .
```

OWL/XML Syntax

```
<ClassAssertion>
   <Class IRI="Woman"/>
   <NamedIndividual IRI="Mary"/>
</ClassAssertion>
```

Class hierarchies

- The classes Woman and Person are related to each other
- Woman is a subclass of Person

Warning about “is a”

- Mary “is a” Woman
- Woman “is a” Person
- Say “is an instance of” or “is a subclass of” instead

Functional-Style Syntax

```
SubClassOf( :Woman :Person )
```

RDF/XML Syntax

```
<owl:Class rdf:about="Woman">
  <rdfs:subClassOf rdf:resource="Person"/>
</owl:Class>
```

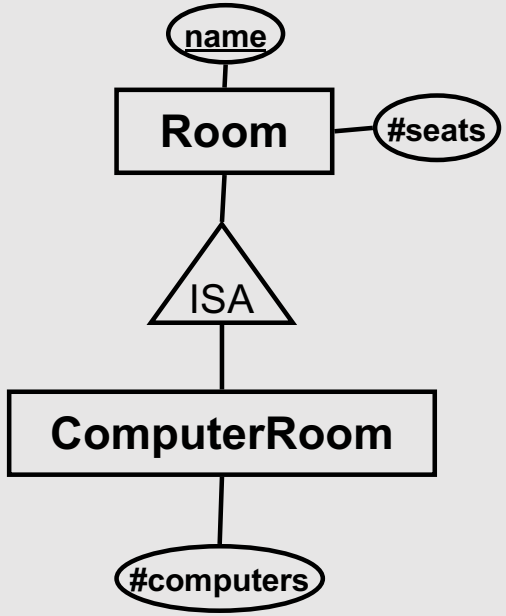
Turtle Syntax

```
:Woman rdfs:subClassOf :Person .
```

OWL/XML Syntax

```
<SubClassOf>
  <Class IRI="Woman"/>
  <Class IRI="Person"/>
</SubClassOf>
```

Question 1 (GK-4)



This E-R diagram can be translated to relations in different ways. Three of these are OK, but which one is not OK?

A.
```
Rooms(name, #seats)
ComputerRooms(name, #computers)
```

B.
```
Rooms(name, #seats)
ComputerRooms(name, #seats, #computers)
```

C.
```
Rooms(name, #seats, #computers)
ComputerRooms(name, #seats, #computers)
```

D.
```
Rooms(name, #seats, #computers)
```

Subclass relationship is transitive

- All mothers are women
- Mother is a subclass of Woman, and also a subclass of Person

- The subclass relationship is *reflexive*
- Every class is its own subclass

Functional-Style Syntax

```
SubClassOf( :Mother :Woman )
```

RDF/XML Syntax

```
<owl:Class rdf:about="Mother">
  <rdfs:subClassOf rdf:resource="Woman"/>
</owl:Class>
```

Turtle Syntax

```
:Mother rdfs:subClassOf :Woman .
```

OWL/XML Syntax

```
<SubClassOf>
  <Class IRI="Mother"/>
  <Class IRI="Woman"/>
</SubClassOf>
```

Equivalent classes

- The terms Person and Human can be used interchangeably

- Person is a subclass of Human
- Human is a subclass of Person

Functional-Style Syntax

```
EquivalentClasses( :Person :Human )
```

RDF/XML Syntax

```
<owl:Class rdf:about="Person">
  <owl:equivalentClass rdf:resource="Human"/>
</owl:Class>
```

Turtle Syntax

```
:Person owl:equivalentClass :Human .
```

OWL/XML Syntax

```
<EquivalentClasses>
  <Class IRI="Person"/>
  <Class IRI="Human"/>
</EquivalentClasses>
```

Disjoint classes

- Can infer that classes Mother and Man are disjoint, and can deduce that Mary is not a man.
- Often omitted
- DisjointClasses(:Person :Car) ?

Functional-Style Syntax

```
DisjointClasses( :Woman :Man )
```

RDF/XML Syntax

```
<owl:AllDisjointClasses>
  <owl:members rdf:parseType="Collection">
    <owl:Class rdf:about="Woman"/>
    <owl:Class rdf:about="Man"/>
  </owl:members>
</owl:AllDisjointClasses>
```

Turtle Syntax

```
[]  rdf:type      owl:AllDisjointClasses ;
    owl:members   ( :Woman   :Man ) .
```

OWL/XML Syntax

```
<DisjointClasses>
    <Class IRI="Woman"/>
    <Class IRI="Man"/>
</DisjointClasses>
```

Relationships between instances

- The order in which individuals are written is important.
- Subject-predicate-object triple is clear in Turtle syntax

Functional-Style Syntax

```
ObjectPropertyAssertion( :hasWife :John :Mary )
```

RDF/XML Syntax

```
<rdf:Description rdf:about="John">
  <hasWife rdf:resource="Mary"/>
</rdf:Description>
```

Turtle Syntax

```
:John :hasWife :Mary .
```

OWL/XML Syntax

```
<ObjectPropertyAssertion>
  <ObjectProperty IRI="hasWife"/>
  <NamedIndividual IRI="John"/>
  <NamedIndividual IRI="Mary"/>
</ObjectPropertyAssertion>
```

Open-world assumption in OWL

- Anything is possible unless you state otherwise
- In contrast, a closed-world assumption is common in databases
- With a closed-world assumption it is not necessary to state all negative cases

Functional-Style Syntax

```
NegativeObjectPropertyAssertion( :hasWife :Bill :Mary )
```

RDF/XML Syntax

```
<owl:NegativePropertyAssertion>
  <owl:sourceIndividual rdf:resource="Bill"/>
  <owl:assertionProperty rdf:resource="hasWife"/>
  <owl:targetIndividual rdf:resource="Mary"/>
</owl:NegativePropertyAssertion>
```

Turtle Syntax

```
[]  rdf:type                owl:NegativePropertyAssertion ;
    owl:sourceIndividual    :Bill ;
    owl:assertionProperty   :hasWife ;
    owl:targetIndividual    :Mary .
```

OWL/XML Syntax

```
<NegativeObjectPropertyAssertion>
  <ObjectProperty IRI="hasWife"/>
  <NamedIndividual IRI="Bill"/>
  <NamedIndividual IRI="Mary"/>
</NegativeObjectPropertyAssertion>
```

Domain and range restrictions

- Restrictions on the types of the subject and object in a triple
- Turtle shorthand for:

```
:hasWife rdfs:domain :Man .
:hasWife rdfs:range  :Woman .
```

Functional-Style Syntax

```
ObjectPropertyDomain( :hasWife :Man )
ObjectPropertyRange( :hasWife :Woman )
```

RDF/XML Syntax

```
<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:domain rdf:resource="Man"/>
  <rdfs:range rdf:resource="Woman"/>
</owl:ObjectProperty>
```

Turtle Syntax

```
:hasWife rdfs:domain :Man ;
         rdfs:range  :Woman .
```

OWL/XML Syntax

```
<ObjectPropertyDomain>
  <ObjectProperty IRI="hasWife"/>
  <Class IRI="Man"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
  <ObjectProperty IRI="hasWife"/>
  <Class IRI="Woman"/>
</ObjectPropertyRange>
```

Question 2 (GK-4)

The domain and range of an object property can be the same.

A. True
B. False

Domain and range restrictions

- Restrictions on the types of the subject and object in a triple

- Turtle shorthand for:

```
:hasWife rdfs:domain :Man .
:hasWife rdfs:range  :Woman .
```

The domain and range of an object property can be the same.

A. True
B. False

Functional-Style Syntax

```
ObjectPropertyDomain( :hasWife :Man )
ObjectPropertyRange( :hasWife :Woman )
```

RDF/XML Syntax

```
<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:domain rdf:resource="Man"/>
  <rdfs:range rdf:resource="Woman"/>
</owl:ObjectProperty>
```

Turtle Syntax

```
:hasWife rdfs:domain :Man ;
         rdfs:range  :Woman .
```

OWL/XML Syntax

```
<ObjectPropertyDomain>
  <ObjectProperty IRI="hasWife"/>
  <Class IRI="Man"/>
</ObjectPropertyDomain>
<ObjectPropertyRange>
  <ObjectProperty IRI="hasWife"/>
  <Class IRI="Woman"/>
</ObjectPropertyRange>
```

Property hierarchies

- If a specific relationship holds between two entities, then any more general relationship also holds

Functional-Style Syntax

```
SubObjectPropertyOf( :hasWife :hasSpouse )
```

RDF/XML Syntax

```
<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:subPropertyOf rdf:resource="hasSpouse"/>
</owl:ObjectProperty>
```

Turtle Syntax

```
:hasWife rdfs:subPropertyOf :hasSpouse .
```

OWL/XML Syntax

```
<SubObjectPropertyOf>
  <ObjectProperty IRI="hasWife"/>
  <ObjectProperty IRI="hasSpouse"/>
</SubObjectPropertyOf>
```

Question 3 (GK-4)

The domain and range of a sub-property will be the same as those of the super-property.

A. True
B. False

Property hierarchies

- If a specific relationship holds between two entities, then any more general relationship also holds

Question 3

The domain and range of a sub-property will be the same as those of the super-property.

A. True
B. False

Functional-Style Syntax

```
SubObjectPropertyOf( :hasWife :hasSpouse )
```

RDF/XML Syntax

```
<owl:ObjectProperty rdf:about="hasWife">
  <rdfs:subPropertyOf rdf:resource="hasSpouse"/>
</owl:ObjectProperty>
```

Turtle Syntax

```
:hasWife rdfs:subPropertyOf :hasSpouse .
```

OWL/XML Syntax

```
<SubObjectPropertyOf>
  <ObjectProperty IRI="hasWife"/>
  <ObjectProperty IRI="hasSpouse"/>
</SubObjectPropertyOf>
```

Same instance

- James and Jim are the same individual

Functional-Style Syntax

```
SameIndividual( :James :Jim )
```

RDF/XML Syntax

```
<rdf:Description rdf:about="James">
  <owl:sameAs rdf:resource="Jim"/>
</rdf:Description>
```

Turtle Syntax

```
:James owl:sameAs :Jim.
```

OWL/XML Syntax

```
<SameIndividual>
  <NamedIndividual IRI="James"/>
  <NamedIndividual IRI="Jim"/>
</SameIndividual>
```

Different instances

- John and Bill are not the same individual
- Listing all different pairs would be a lot of work!

Functional-Style Syntax

```
DifferentIndividuals( :John :Bill )
```

RDF/XML Syntax

```
<rdf:Description rdf:about="John">
  <owl:differentFrom rdf:resource="Bill"/>
</rdf:Description>
```

Turtle Syntax

```
:John owl:differentFrom :Bill .
```

OWL/XML Syntax

```
<DifferentIndividuals>
  <NamedIndividual IRI="John"/>
  <NamedIndividual IRI="Bill"/>
</DifferentIndividuals>
```

Scalar-valued property

Functional-Style Syntax

```
DataPropertyAssertion( :hasAge :John "51"^^xsd:integer )
```

RDF/XML Syntax

```
<Person rdf:about="John">
  <hasAge rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">51</hasAge>
</Person>
```

Turtle Syntax

```
:John  :hasAge  51 .
```

OWL/XML Syntax

```
<DataPropertyAssertion>
  <DataProperty IRI="hasAge"/>
  <NamedIndividual IRI="John"/>
  <Literal datatypeIRI="http://www.w3.org/2001/XMLSchema#integer">51</Literal>
</DataPropertyAssertion>
```

Inverse relationships

- The property hasParent can be defined as the inverse property of hasChild

Functional-Style Syntax

```
InverseObjectProperties( :hasParent :hasChild )
```

RDF/XML Syntax

```
<owl:ObjectProperty rdf:about="hasParent">
  <owl:inverseOf rdf:resource="hasChild"/>
</owl:ObjectProperty>
```

Turtle Syntax

```
:hasParent owl:inverseOf :hasChild .
```

OWL/XML Syntax

```
<InverseObjectProperties>
  <ObjectProperty IRI="hasParent"/>
  <ObjectProperty IRI="hasChild"/>
</InverseObjectProperties>
```

Question 4 (GK-4)

An inverse property must be declared for every object property.

A. True
B. False

Inverse relationships

- The property hasParent can be defined as the inverse property of hasChild

Question 4

An inverse property must be declared for every object property.

A. True
B. False

Functional-Style Syntax

```
InverseObjectProperties( :hasParent :hasChild )
```

RDF/XML Syntax

```
<owl:ObjectProperty rdf:about="hasParent">
  <owl:inverseOf rdf:resource="hasChild"/>
</owl:ObjectProperty>
```

Turtle Syntax

```
:hasParent owl:inverseOf :hasChild .
```

OWL/XML Syntax

```
<InverseObjectProperties>
  <ObjectProperty IRI="hasParent"/>
  <ObjectProperty IRI="hasChild"/>
</InverseObjectProperties>
```

Protégé

- An ontology editor
- Developed at Stanford University
- ”A Practical Guide To Building OWL Ontologies Using Protege 4”
  - http://mowl-power.cs.man.ac.uk/protegeowltutorial/resources/ProtegeOWLTutorialP4_v1_3.pdf

- https://protege.stanford.edu/

Who uses ontologies?

- There are many ontologies, of various sizes, being used across many sectors.

- Some are listed on the next few slides to give a flavour of where ontologies are being used.

Schema.org

(Featured in Hendler's talk from [16:13])

- Founded by Google, Microsoft, Yahoo and Yandex
- http://schema.org/
- Used by over 10 million web sites
- "A shared vocabulary makes it easier for webmasters and developers to decide on a schema and get the maximum benefit for their efforts. It is in this spirit that the founders, together with the larger community have come together - to provide a shared collection of schemas. "

DBpedia ontology

- https://wiki.dbpedia.org/services-resources/ontology

- 685 classes; 2795 properties; 4,233,000 instances

- Directed acyclic graph (not a tree) – this is similar to the Geno Ontology (see leter)

BBC ontologies

- https://www.bbc.co.uk/ontologies

- The ontologies can be downloaded as RDF files in Turtle format, and can be loaded into Protégé.

Ontologies and the built environment

CityGML 2.0 in OWL:
http://vgibox.eu/repository/index.php/CityGML_in_OWL

OpenStreetMap in OWL:
https://wiki.openstreetmap.org/wiki/OSMonto

Industry Foundation Classes in OWL:
https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/

The Gene Ontology

- “The mission of the GO Consortium is to develop a comprehensive, **computational model of biological systems**, ranging from the molecular to the organism level, across the multiplicity of species in the tree of life.

  The Gene Ontology (GO) knowledgebase is the world’s largest source of information on the functions of genes. This knowledge is both human-readable and machine-readable, and is a foundation for computational analysis of large-scale molecular biology and genetics experiments in biomedical research.“

- http://geneontology.org/

The Gene Ontology

Three main aspects:

- molecular function
- cellular component
- biological function

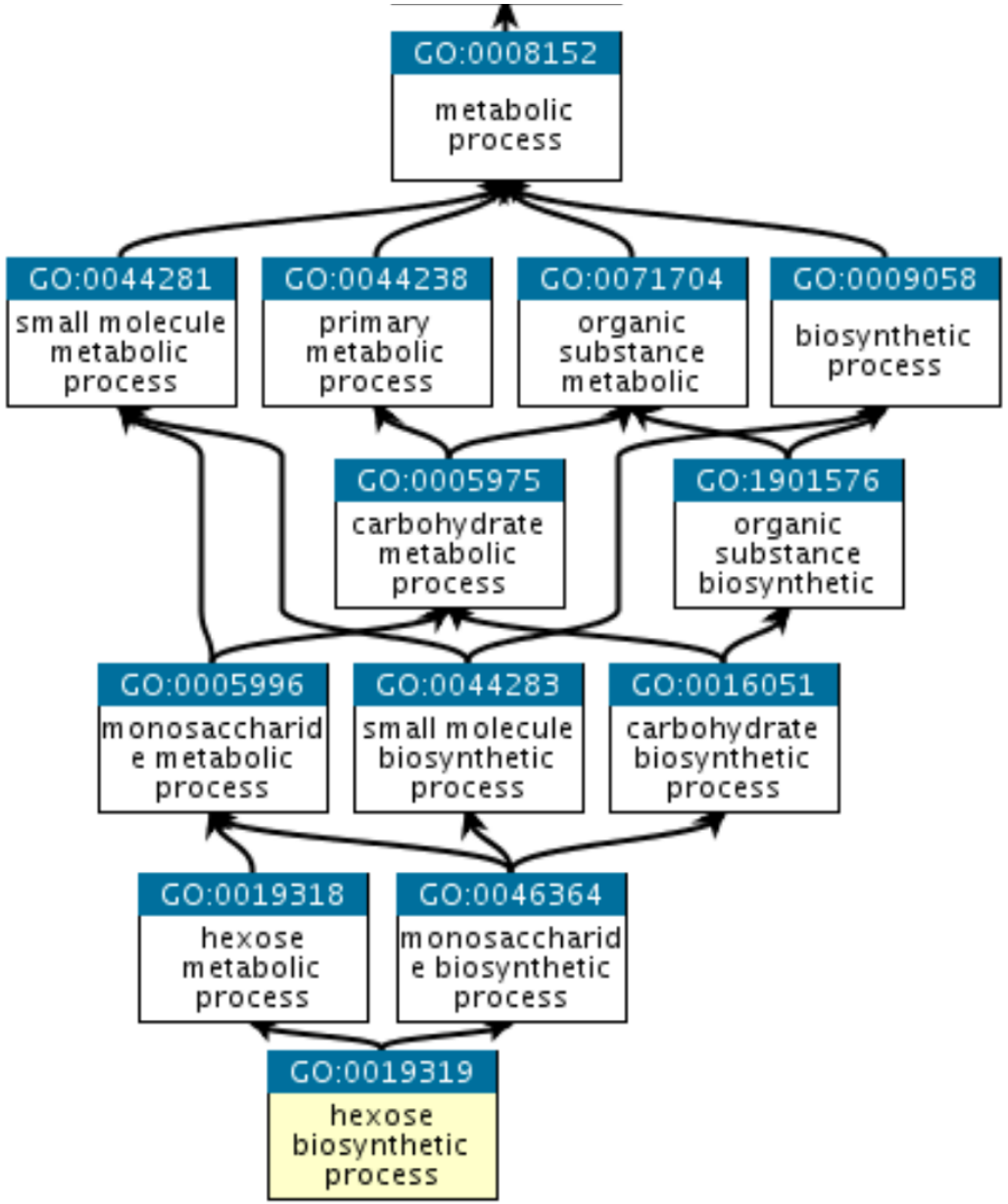
commonly used relationships in GO include:

- *is a* (is a subtype of)
- *part of* (if “B part of A “ then whenever B exists it is part of A)
- has part (if “A has part B” then if A exists then B exists)
- *regulates*
    - *negatively regulates*
    - *positively regulates*

Directed acyclic graph

- Shows how different terms or phrases are related to each another.
- Not a good design for a database!

http://geneontology.org/docs/ontology-documentation/



The Gene Ontology and the SARS-CoV-2 virus

- http://geneontology.org/covid-19.html

- functions of the two human proteins used by the virus to enter a human cell

- functions of human proteins possibly targeted by the virus after cell entry
  - the viral genome encodes 29 proteins
  - 27 of these have been tested to see with which human proteins they can physically interact
  - 332 potentially interacting human proteins identified, including 67 that are targeted by existing FDA-approved drugs

Biomedical ontologies portal

- 850 biomedical ontologies
- https://bioportal.bioontology.org/

Build your own?

In an introductory database course you will typically design, construct and use your own database using a (relational) database management system. You will define the structure of the database (by creating tables), insert data into the database (using SQL statements) and then query the data (using the SQL query sub-language).

We've seen how to query Semantic Web data in existing data resources like DBpedia and Wikidata (using the SPARQL language), but what about designing, creating and using semantic graphs of our own?

We can use a database management system like GraphDB.

We can use Protégé to define a schema for a semantic graph database

- Can define the classes (arranged into class hierarchies), relationships with their domains and ranges, and attributes.
- Similar to defining an E-R model.
- Can save as XML-RDF, then load this into GraphDB.

- Do not try to include all terms in the domain of interest (see slide on directed acyclic graphs in the Gene Ontology for why this would be a bad idea).

GraphDB

- A semantic graph database

- https://ontotext.com/products/graphdb/

Sharing data on the Web

- Plain text
- HTML
- XML
- XML + DTD / XML Schema
- RDF
- RDF Schema
- OWL Lite
- OWL Full

Querying

- Xpath and Xquery
- SPARQL

Model

- Basically trees
- Basically graphs

Preparation for class on Wednesday 13 May

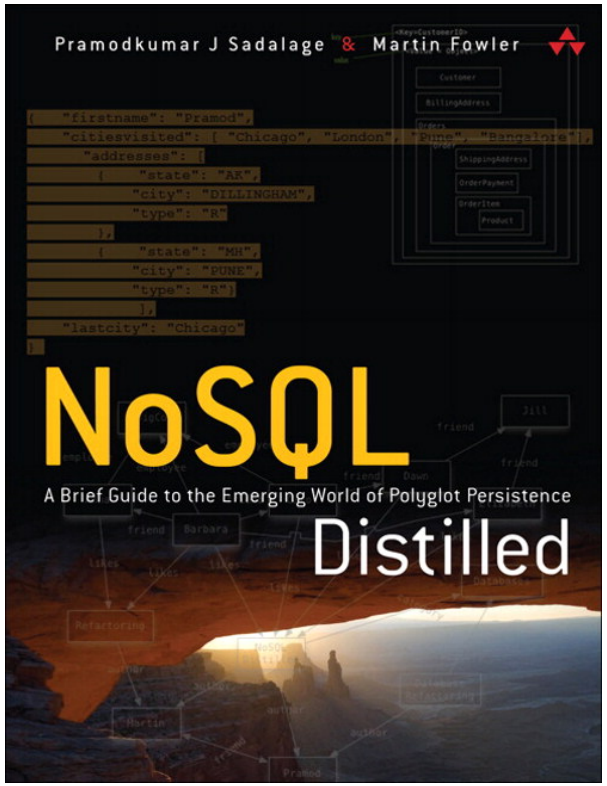
- Pramod J Sadalage and Martin Fowler have distilled the most important concepts and issues around NoSQL databases into a book

- Martin Fowler has further distilled these concepts and issues into a lecture

  GOTO 2012

  - Introduction to NoSQL
  - Martin Fowler

- https://www.youtube.com/watch?v=qI_g07C_Q5I



Some questions on Martin Fowler's lecture

- What is the ***impedence mismatch problem***?
- What are the common characteristics of NoSQL databases?
- What is an ***implicit schema***?
- Describe each of the four main data models seen in NoSQL databases.
- What are ***sharding*** and ***replication***?
- Give two main reasons why developers move to using a NoSQL database.
- What is ***polyglot persistence***?

Vacancies

- Teaching assistant, one or more (Data Science and AI), PAR 2020/555
- Autumn semester 2020

- Closing date: 2020-05-24

https://www.chalmers.se/en/about-chalmers/Working-at-Chalmers/Vacancies/Pages/default.aspx?rmpage=job&rmjob=8579&rmlang=UK