COMP9321:
Data services engineering

Semester 1, 2018,

Week 7 Accessing and Publishing Data (Part 2)

By Helen Paik, CSE UNSW

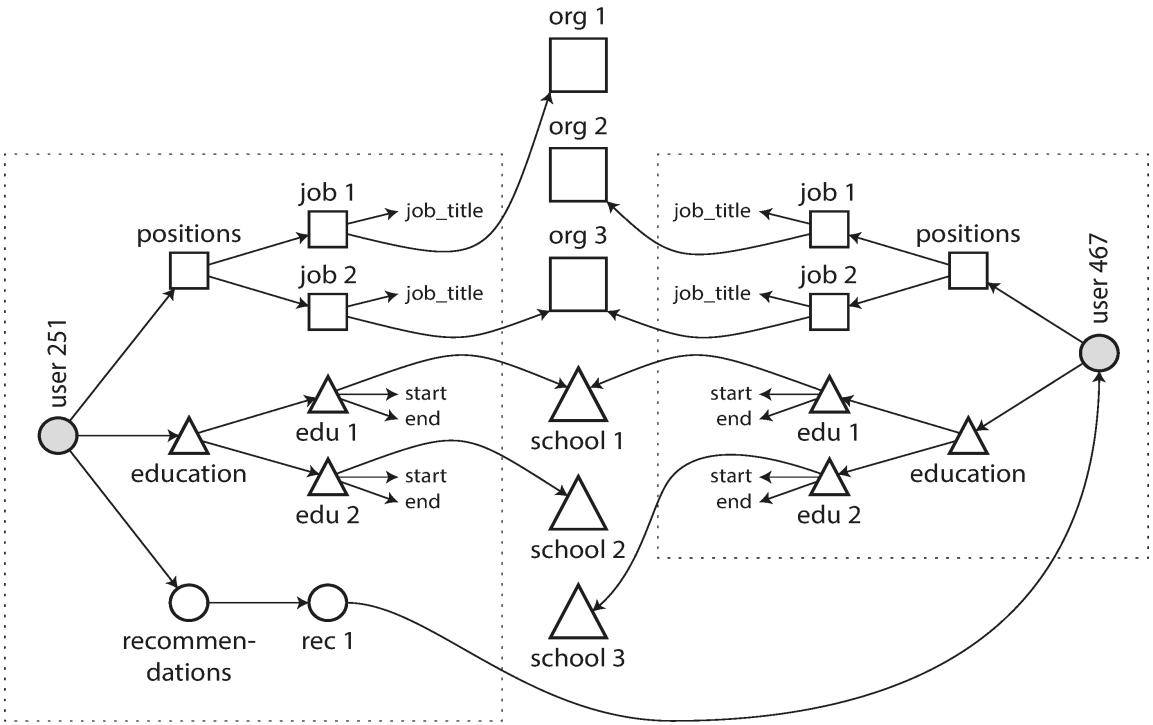
Graph-like Models

M-M relationships are an important factor in deciding which data model to go with

1-M (tree/doc), self-contained -> Document model

M-M -> either relational or graph

Highly M-M, complicated connections -> graph …



The recommendations – linking it to other Users **(Many-to-Many Relationships)**

Graph:

- Vertices/nodes: represent entities
- Edges/arcs: represent relationships

Graph-like Models

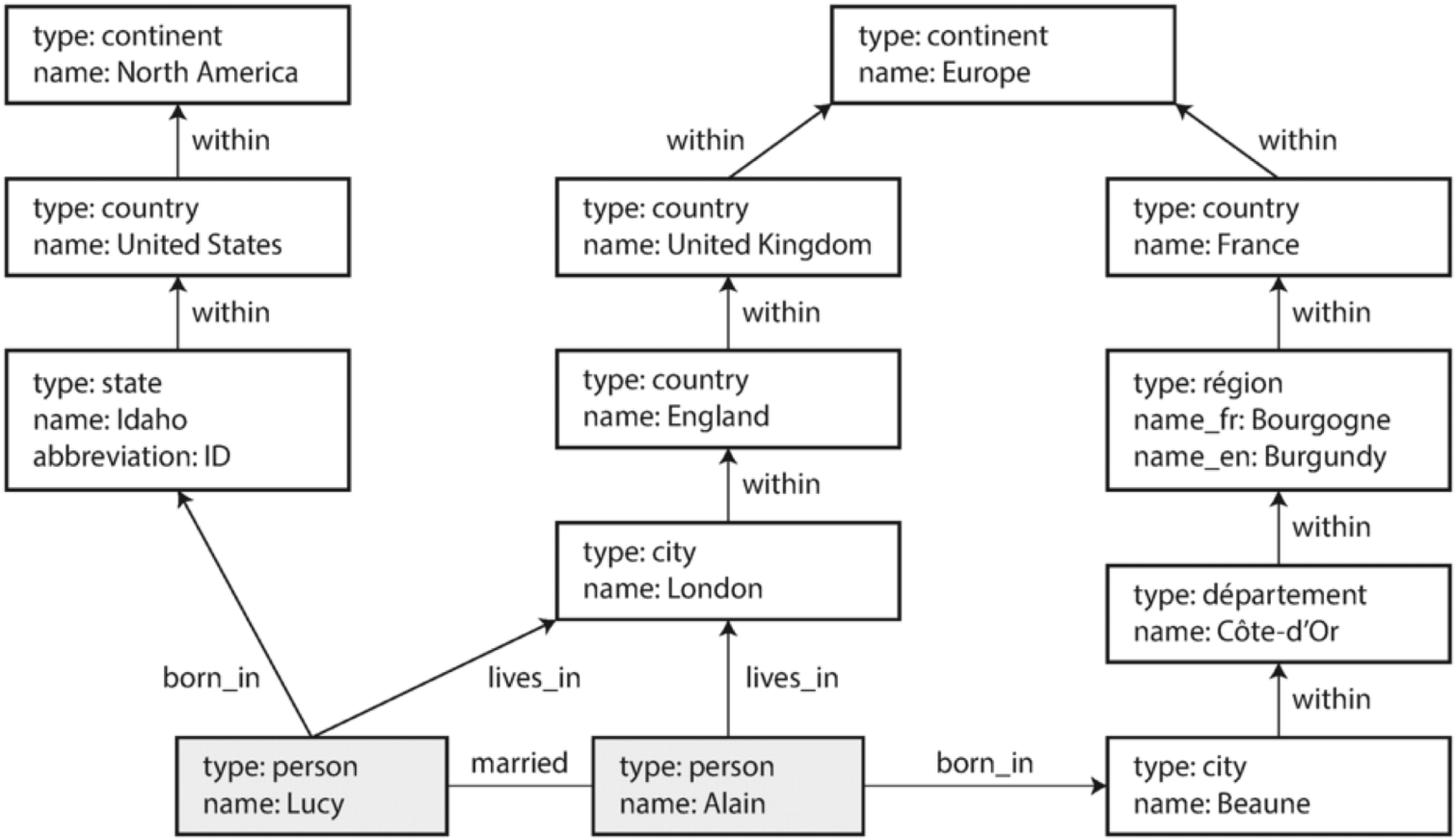
Many kinds of data can be modelled as a graph

- Social Graph – vertices are people, edges indicate which people know each other
- The Web Graph – vertices are web pages and edges indicate HTML links to other pages
- Road or Rail networks – vertices are junctions and edges represent the roads/railways between them

Well-known algorithms on the model

http://www.supplychain247.com/article/why_supply_chains_should_be_more_socially_engaged
http://canacopegdl.com/single.php?id=http://www-inst.eecs.berkeley.edu/~cs61bl/r//cur/graphs/web.graph.png
https://visualign.wordpress.com/2012/07/11/london-tube-map-and-graph-visualizations/

Graph-like Models



Vertices are not limited to the same type of data.

Graph-like Models

Facebook, TAO system (2013)

a)

b)

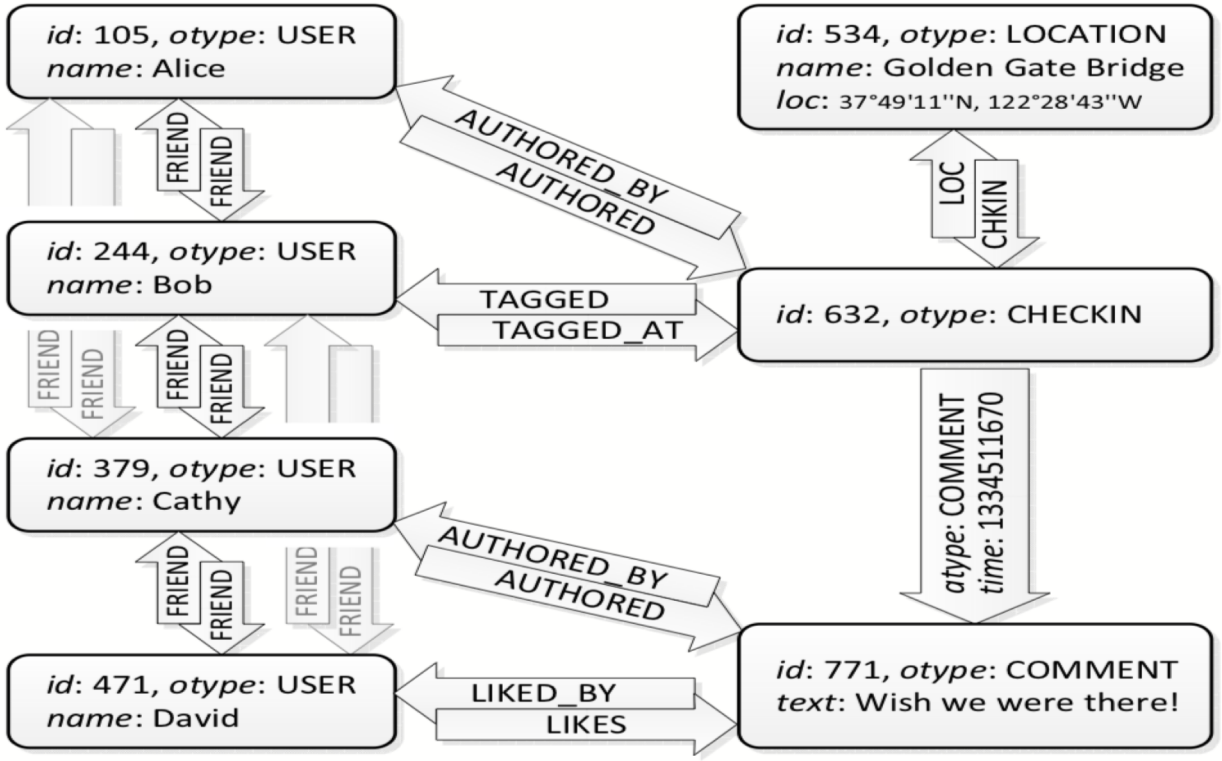


Figure 1: A running example of how a user's checkin might be mapped to objects and associations.

Designing Data-Intensive Applications, by Martin Kleppmann
(https://www.usenix.org/system/files/conference/atc13/atc13-bronson.pdf)

Storing and Querying Graph-like Models

Property Graph model:

Each vertex:

- Identifier
- A set of outgoing edges
- A set of incoming edges
- A collection of properties (key-value pairs)

Each edge:

- Identifier
- The vertex at which the edge starts (tail)
- The vertex at which the edge ends (head)
- A label for the relationship
- A collection of properties

(e.g., PostgreSQL, using json type)

Example 2-2. Representing a property graph using a relational schema

```
CREATE TABLE vertices (
    vertex_id   integer PRIMARY KEY,
    properties  json
);

CREATE TABLE edges (
    edge_id     integer PRIMARY KEY,
    tail_vertex integer REFERENCES vertices (vertex_id),
    head_vertex integer REFERENCES vertices (vertex_id),
    label       text,
    properties  json
);

CREATE INDEX edges_tails ON edges (tail_vertex);
CREATE INDEX edges_heads ON edges (head_vertex);
```

Storing and Querying Graph-like Models

Property Graph model:

Any vertex can have edges (no schema-based restriction on what kinds of 'things' can be connected)

Given any vertex, you can efficiently find both incoming and outcoming edges – traversing the graph

By using different labels for different types of relationships, you can store several different kinds of information in a single graph

These features give graphs a great flexibility for data modelling

(e.g., PostgreSQL, using json type)

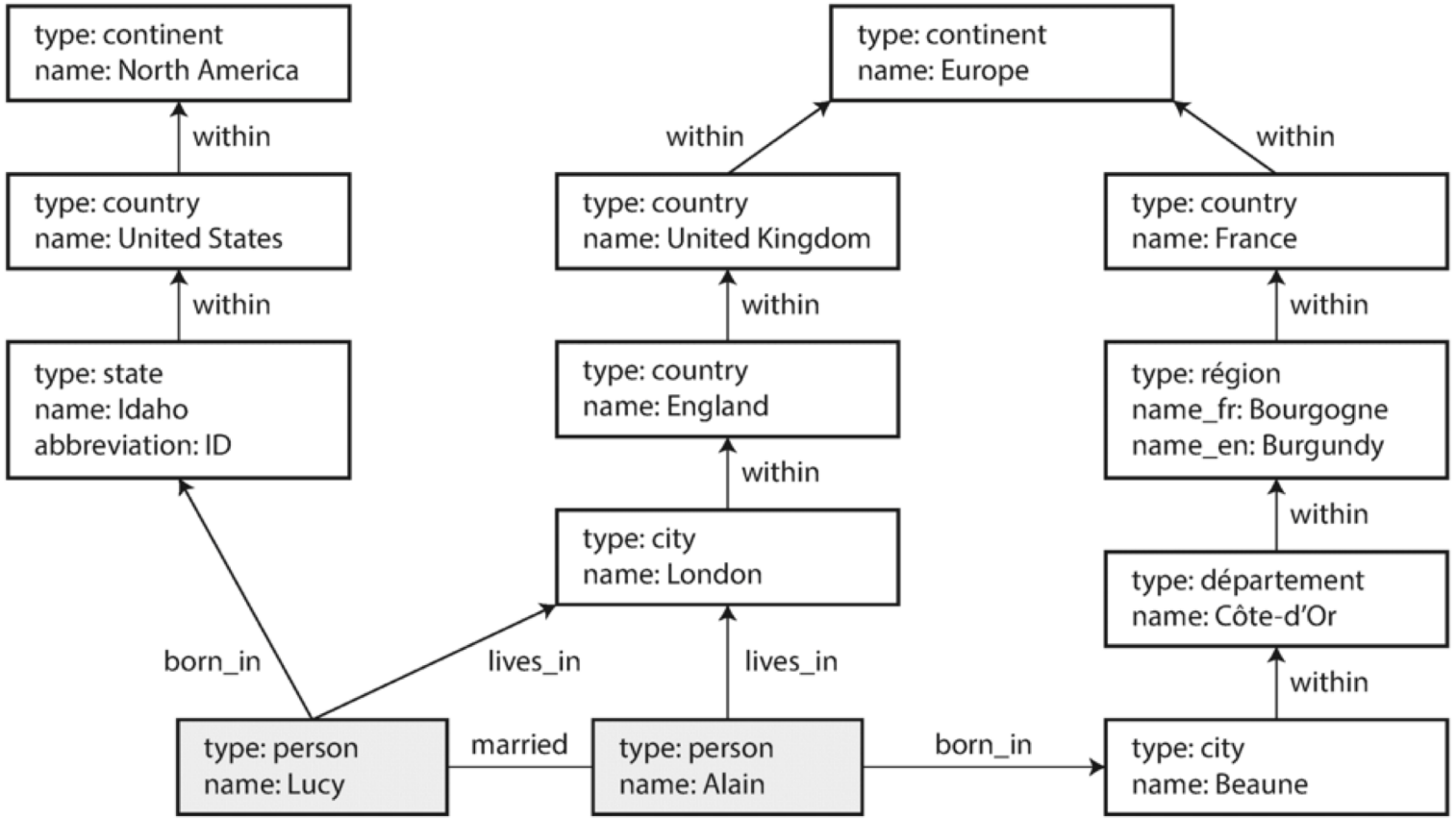
Example 2-2. Representing a property graph using a relational schema

```
CREATE TABLE vertices (
    vertex_id   integer PRIMARY KEY,
    properties  json
);

CREATE TABLE edges (
    edge_id     integer PRIMARY KEY,
    tail_vertex integer REFERENCES vertices (vertex_id),
    head_vertex integer REFERENCES vertices (vertex_id),
    label       text,
    properties  json
);

CREATE INDEX edges_tails ON edges (tail_vertex);
CREATE INDEX edges_heads ON edges (head_vertex);
```

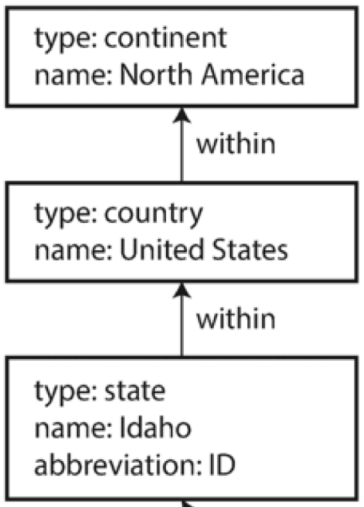
So graphs are “very” flexible … (cf. RDB)



- Different kinds of regional structures in different countries
- Type country “within” a type country
- Varying granularity (e.g., born_in “type:state”, lives_in type:city)

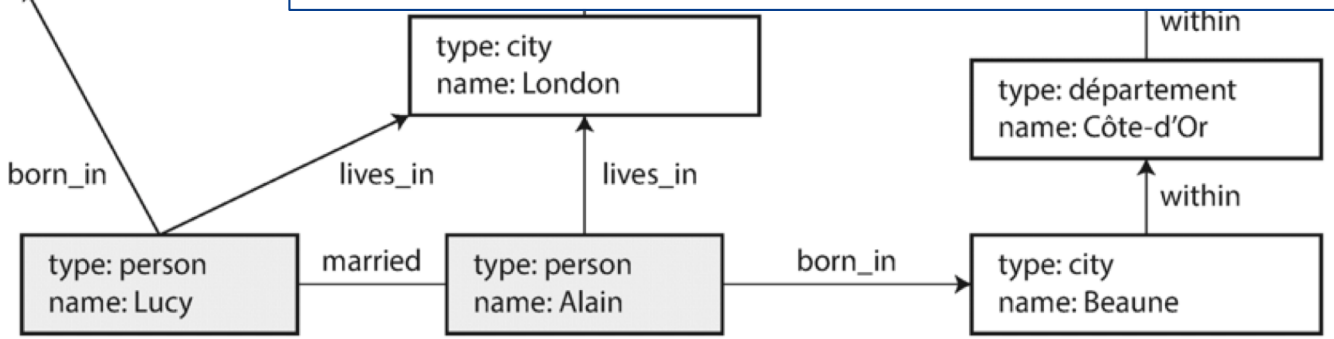
Storing and Querying Graph-like Models

Cypher Query - declarative query language for graphs



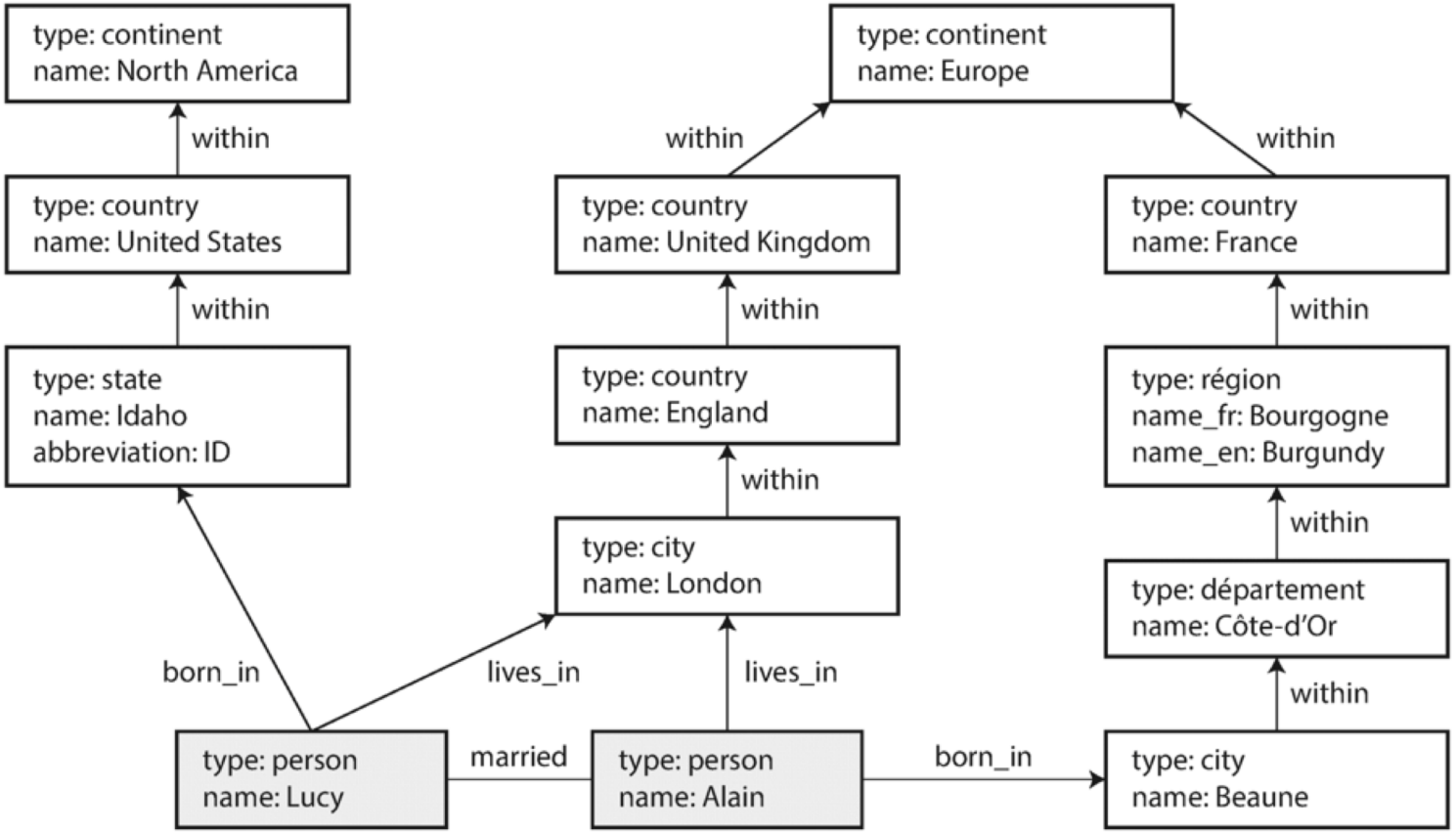
Example 2-3. A subset of the data in Figure 2-5, represented as a Cypher query

```
CREATE
  (NAmerica:Location {name:'North America', type:'continent'}),
  (USA:Location      {name:'United States', type:'country'  }),
  (Idaho:Location    {name:'Idaho',         type:'state'    }),
  (Lucy:Person       {name:'Lucy' }),
  (Idaho) -[:WITHIN]->  (USA)  -[:WITHIN]-> (NAmerica),
  (Lucy)  -[:BORN_IN]-> (Idaho)
```



Example 2-4. Cypher query to find people who emigrated from the US to Europe

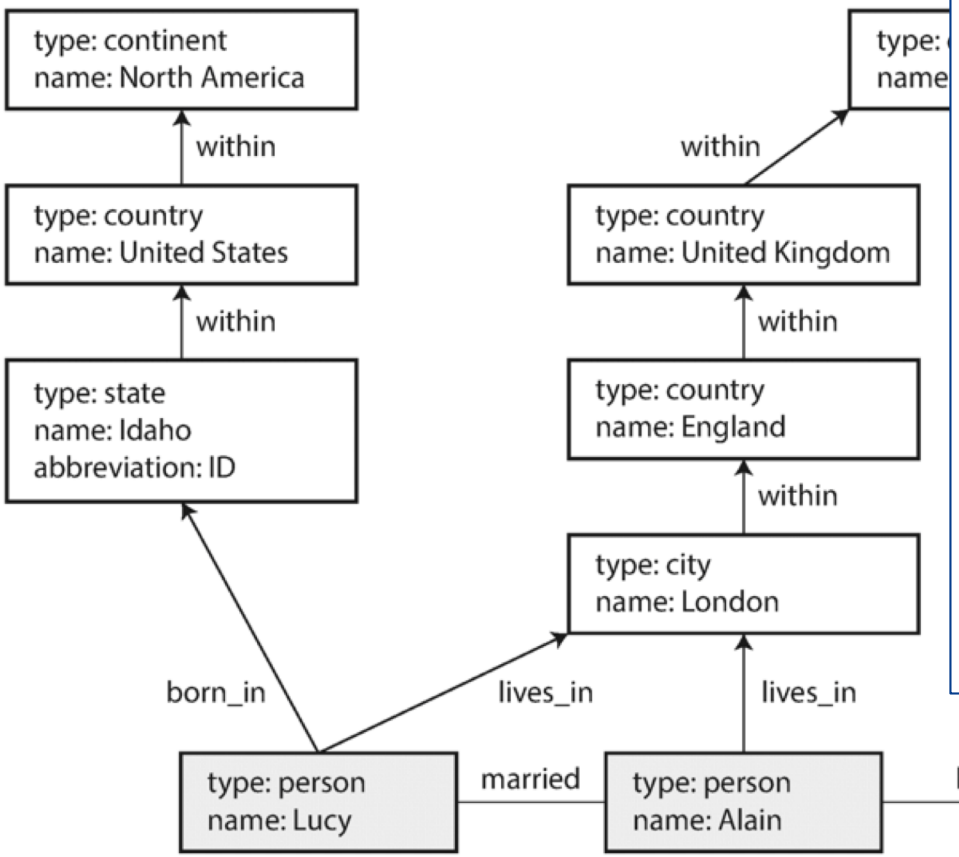
```
MATCH
  (person) -[:BORN_IN]->  () -[:WITHIN*0..]-> (us:Location {name:'United
States'}),
  (person) -[:LIVES_IN]-> () -[:WITHIN*0..]-> (eu:Location
{name:'Europe'})
RETURN person.name
```



"Find the names of all the people who emigrated from the United States to Europe"

"Declarative" language -> the execution details hidden

Doing the same in relational tables and SQL … ??



Example 2-2. Representing a property graph using a relational schema

```
CREATE TABLE vertices (
    vertex_id   integer PRIMARY KEY,
    properties  json
);

CREATE TABLE edges (
    edge_id     integer PRIMARY KEY,
    tail_vertex integer REFERENCES vertices (vertex_id),
    head_vertex integer REFERENCES vertices (vertex_id),
    label       text,
    properties  json
);

CREATE INDEX edges_tails ON edges (tail_vertex);
CREATE INDEX edges_heads ON edges (head_vertex);
```

```
WITH RECURSIVE

  -- in_usa is the set of vertex IDs of all locations within
the United States
  in_usa(vertex_id) AS (
      SELECT vertex_id FROM vertices WHERE properties->>'name'
= 'United States' ❶
    UNION
      SELECT edges.tail_vertex FROM edges ❷
        JOIN in_usa ON edges.head_vertex = in_usa.vertex_id
        WHERE edges.label = 'within'
  ),

  -- in_europe is the set of vertex IDs of all locations within
Europe
  in_europe(vertex_id) AS (
      SELECT vertex_id FROM vertices WHERE properties->>'name'
= 'Europe' ❸
    UNION
      SELECT edges.tail_vertex FROM edges
        JOIN in_europe ON edges.head_vertex =
in_europe.vertex_id
        WHERE edges.label = 'within'
  ),

  -- born_in_usa is the set of vertex IDs of all people born in
the US
  born_in_usa(vertex_id) AS ( ❹
    SELECT edges.tail_vertex FROM edges
      JOIN in_usa ON edges.head_vertex = in_usa.vertex_id
      WHERE edges.label = 'born_in'
  ),

  -- lives_in_europe is the set of vertex IDs of all people
living in Europe
  lives_in_europe(vertex_id) AS ( ❺
    SELECT edges.tail_vertex FROM edges
      JOIN in_europe ON edges.head_vertex = in_europe.vertex_id
      WHERE edges.label = 'lives_in'
  )

SELECT vertices.properties->>'name'
FROM vertices
-- join to find those people who were both born in the US *and*
live in Europe
JOIN born_in_usa     ON vertices.vertex_id =
born_in_usa.vertex_id ❻
JOIN lives_in_europe ON vertices.vertex_id =
lives_in_europe.vertex_id;
```

XML data model and query language

in HTML ...

```
<html>
<h1>Bibliography</h1>
<ol>
<li><i>Foundation of Databases</i>,
  <b>Abiteboul, Hull</b>, 1995</li>
<li><i>Database Systems</i>
  <b>Elmasri, Navathe</b>, 1994</li>
</ol>
</html>
```

in XML ...

```
<bibliography>
<book>
 <title>Foundation of Databases</titl
 <author>Abiteboul</author>
 <author>Hull</author>
 <year>1995</year>
</book>
<book> <!-- continues --> </book>
</bibliography>
```

- A simple, very flexible and extensible text data format
- “extensible” because the markup format is not fixed like HTML
  - It lets you design your own customised markup
- XML is a language that describes data
  - It separates presentation issues from the actual data

XML data model and query language

Consider the following snippet of information from a staff list:

LName	Title	FName	School	Campus	Room
Edgar	Miss	Pam	Optometry	KG	B501
Edmond	Dr	David	Information Systems	GP	S842
Edmonds	Dr	Ian	Physical Sciences	GP	M206

```
<?xml version="1.0"?>
<Phonebook>
   <Entry>
      <LastName Title="Miss">Edgar</LastName>
      <FirstName>Pam</FirstName>
      <School>Optometry</School>
      <Campus>GP</Campus>
      <Room>B501</Room>
      <Extension>5695</Extension>
   </Entry>
   <Entry>
      <LastName Title="Dr">Edmond</LastName>
      <FirstName>David</FirstName>
      <School>Information Systems</School>
      <Campus>GP</Campus>
      <Room>S842</Room>
      <Extension>2240</Extension>
   </Entry> <!— snipped -->
</Phonebook>
```

It's a flexible data representation/model

XML data model and query language

Benefits of using XML in document – especially for data exchange

- Self-describing, modular and portable data
- A common, widely accepted data representation language
- Standard support available for creating/parsing XML docs
- Standard support for checking validity of data
- Efficient search of business information
  - Standard support for querying XML docs
  - Quick and simple search (XPath)
  - More comprehensive keyword + structure based search possible as well (XQuery)
- Extensible document descriptions
  - XML is flexible (cf. relational tables)!
  - reuse, adaptation of existing documents

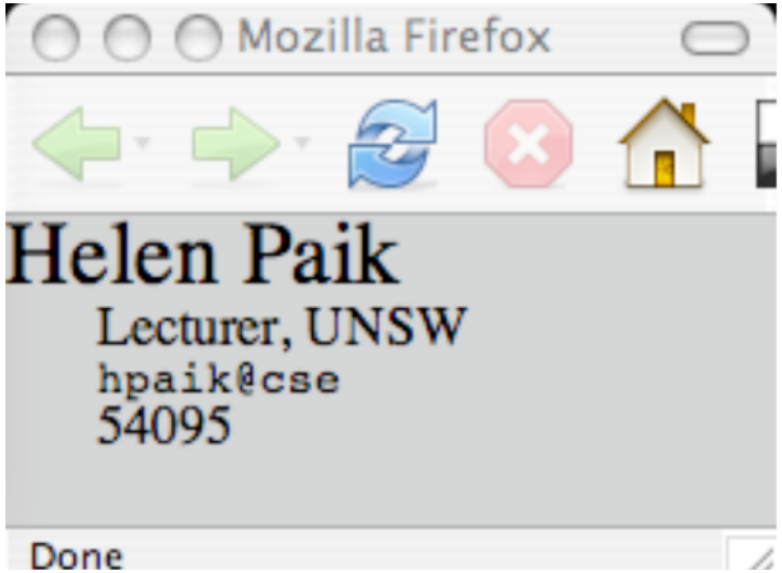
XML – separating content/presentation

XML

```
<?xml version="1.0" ?>
<?xml-stylesheet type="text/css" href="staffcard.css" ?>
<staff>
    <name>Helen Paik</name>
    <title>Lecturer, UNSW</title>
    <email>hpaik@cse</email>
    <extension>54095</extension>
    <photo src="me.gif" />
</staff>
```

CSS

```
staff{background-color: #cccccc; ...}
name{display: block; font-size: 20pt; ... }
title{display: block; margin-left: 20pt;}
email{display: block; font-family: monospace;
extension{display: block; margin-left: 20pt;}
```



XML – separating content/presentation

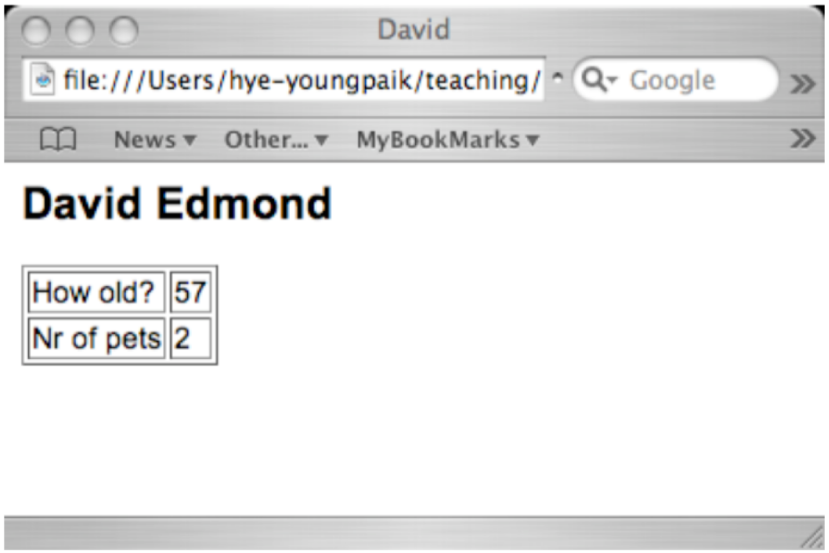
XSLT (Style Sheet Language for XML)

From data-oriented XML:

```
<person>
 <name>
   <given>David</given>
   <family>Edmond</family>
 </name>
 <age>57</age>
 <pets>
   <dog>Winnie</dog>
   <cat>Misty</cat>
 </pets>
</person>
```

file: edmond.xml

You want to produce:



That is:

- extract name, age and # of pets
- display the info in HTML

XML – separating content/presentation

```
<?xml version="1.0"?>
<xsl:stylesheet  xmlns:xsl=http://www.w3.org/1999/XSL/Transform version="1.1">
<xsl:output method="html" indent="no"/>
<xsl:strip-space elements="*"/>
<xsl:template match="/">
<html>
<head>
   <title><xsl:value-of select="person/name/given"/></title>
</head>
<body>
   <h2><xsl:value-of select="person/name/given"/>  
          <xsl:value-of select="person/name/family"/></h2>
<table border="1">
<tr>
    <td>How old?</td>
    <td><xsl:value-of select="person/age"/>
    </td></tr>
<tr>
    <td>Nr of pets</td>
    <td><xsl:value-of select="count(person/pets/child::node())"/>
    </td></tr>
</table>
</body></html>
</xsl:template>
</xsl:stylesheet>
```

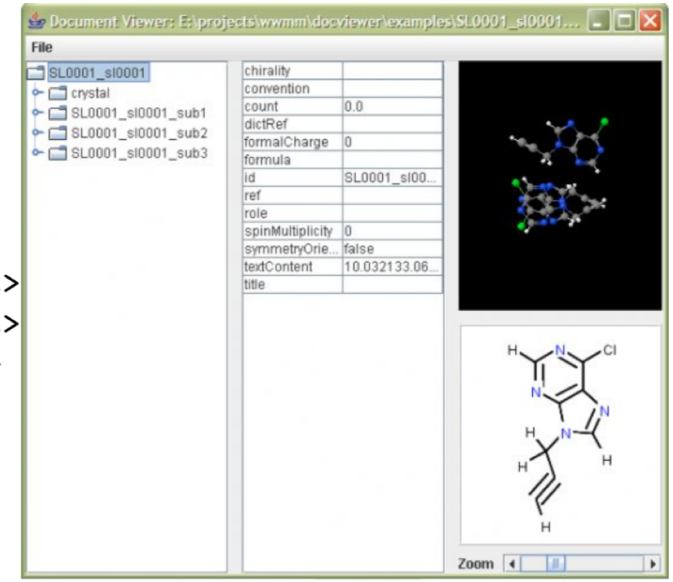
Note – the language itself is in XML …

file: edmond.xsl

XML – many applications …

Chemical Markup Language (CML)

```
<atom id="caffeine_karne_a_1">
 <float builtin="x3" units="A">-2.8709</float>
 <float builtin="y3" units="A">-1.0499</float>
 <float builtin="z3" units="A">0.1718</float>
 <string builtin="elementType">C</string>
</atom>
```



Math Markup Language (MathML)

```
<mrow>
<apply><eq/>
 <ci>A</ci>
 <matrix>
   <matrixrow><ci>x</ci><ci>y</ci></matrixrow>
   <matrixrow><ci>z</ci><ci>w</ci></matrixrow>
 </matrix>
</apply>
</mrow>
```

$$A = \begin{pmatrix} X & Y \\ Z & W \end{pmatrix}$$

XML – many applications …

Data Feeds (RSS and ATOM)

```
<rss version="0.91">
<channel>
<title>CNN.com</title>
<item>
<title>July ends with 76 ... killed</title>
<link>http://www.cnn.com/.../story.html</link>
<description>Three U.S. soldiers were ...</description>
</item>
```

SVG

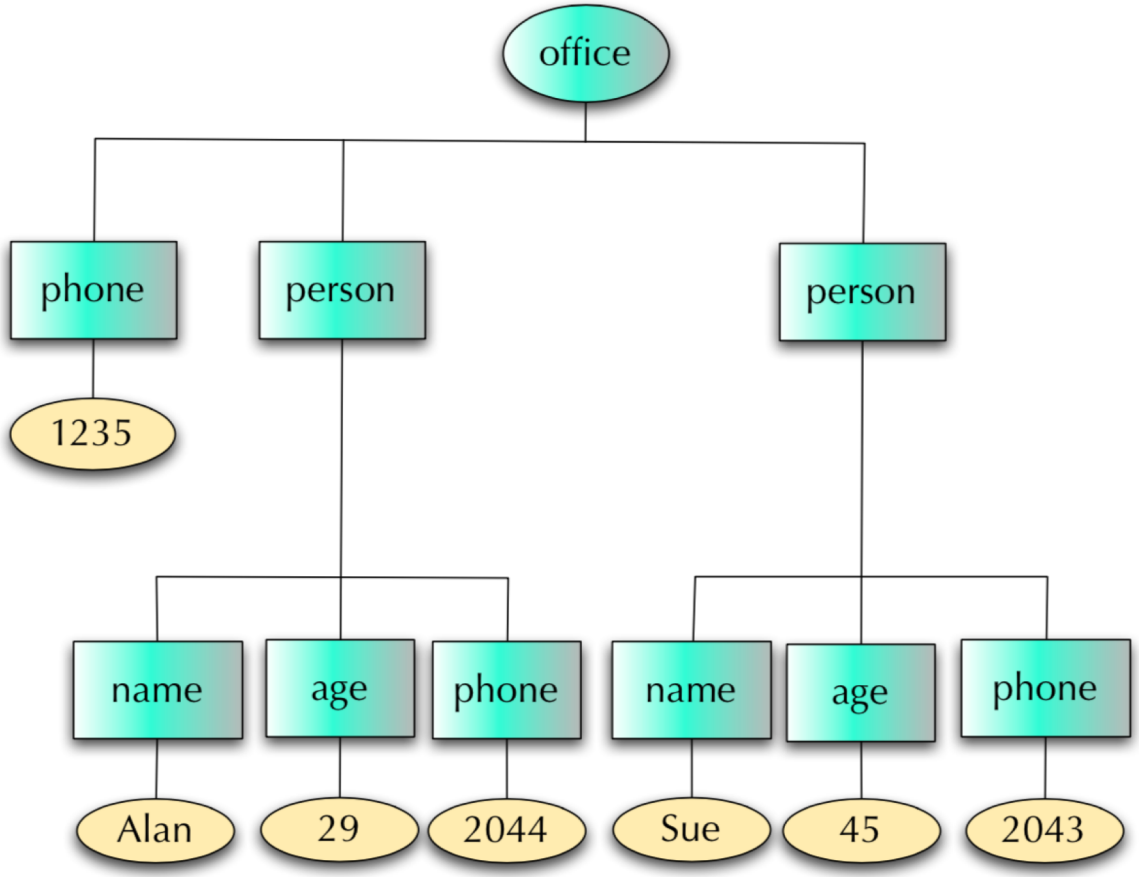
- https://pixabay.com/en/photos/svg/

Many more … (e.g., system configuration files, XML-based APIs)

XML is strongly validated …

XML document is a tree …

```
<office>
   <phone>1235</phone>
   <person>
      <name>Alan</name>
      <age>29</age>
      <phone>2044</phone>
  </person>
  <person>
     <name>Sue</name>
     <age>45</age>
     <phone>2043</phone>
  </person>
</office>
```



XML is strongly validated …

- All XML documents must have ’a’ root element
- All XML elements must have a closing tag Empty element tags end with / >
- XML tags are case sensitive (NAME vs. Name)
- All XML elements must be nested (<q> </q> ??)
- Element Naming
  - letters, numbers, and other characters
  - must not start with a number, ’. (period)’ or ’- (hyphen)’
  - must not start with ’xml’ (or XML or Xml ..)
  - cannot contain spaces
- Attribute values must always be quoted (single or double)
- XML could have a fixed schema (XML Schema, XML DTD)

Namespaces in XML

XML elements can have any names.

What if a name could mean two different things (i.e., name clash)?

From University X:

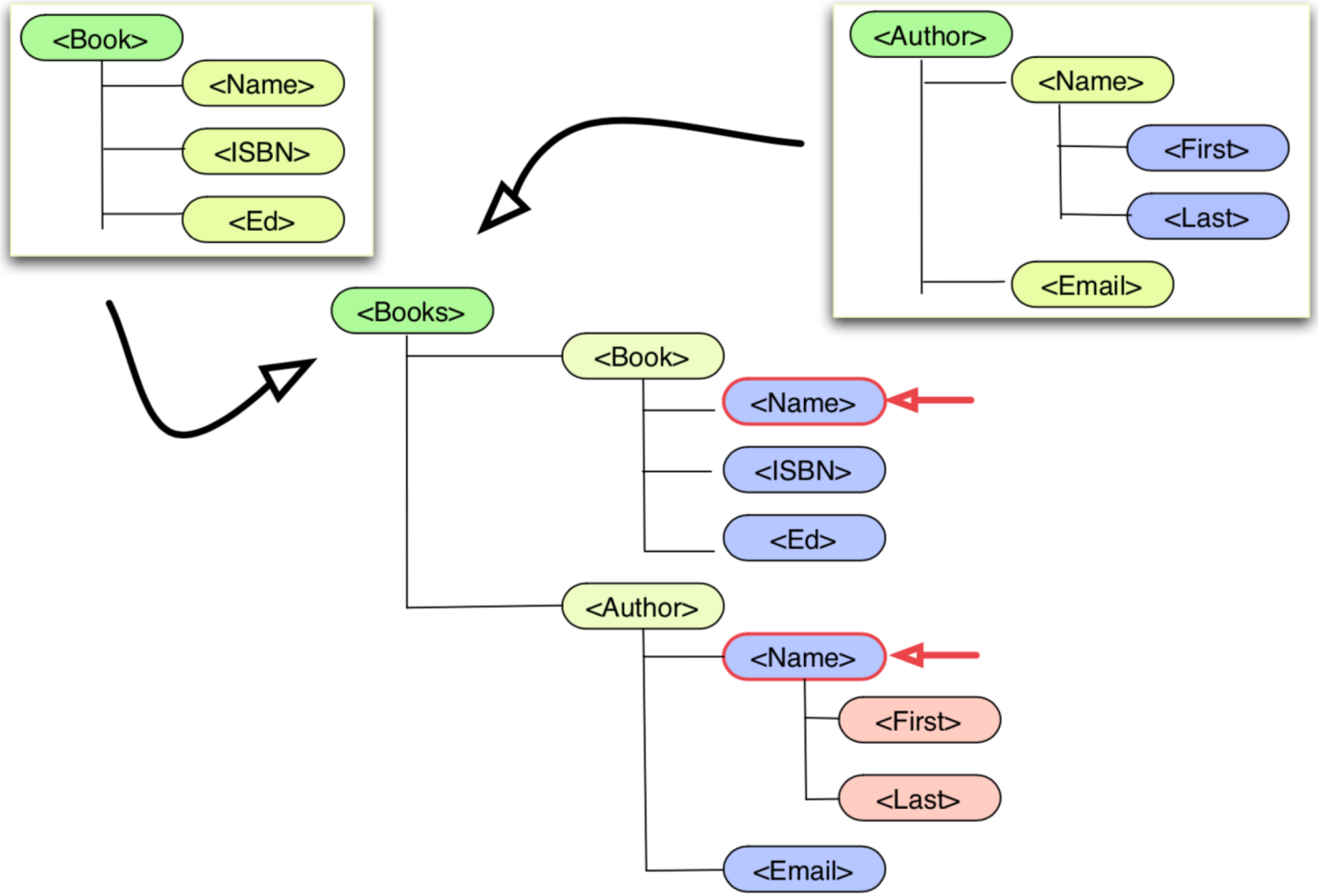
```
<student>
 <id>12345</id>
 <name>Jeff Smith</name>
 <language>C#</language>
 <rating>9.5</rating>
</student>
```

From University Y:

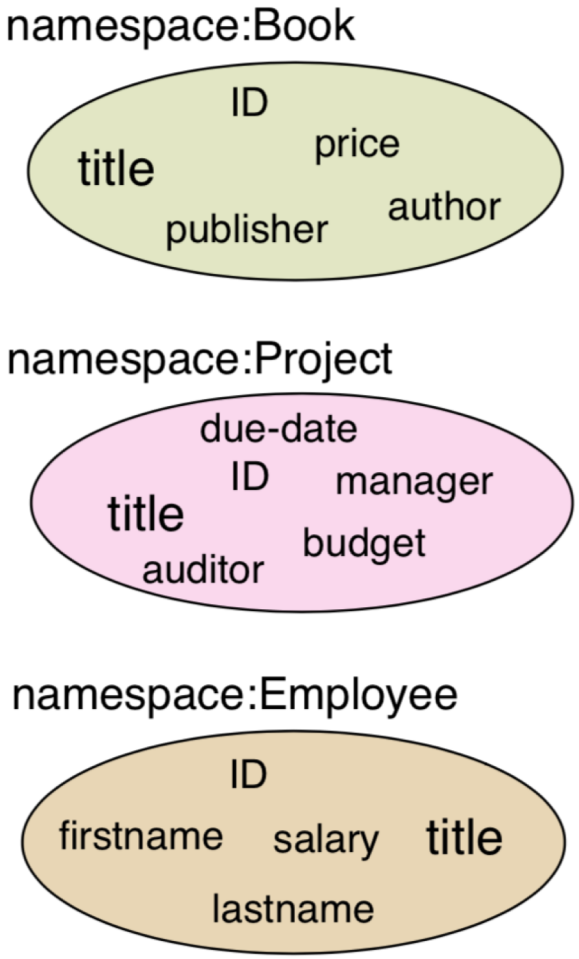
```
<student>
 <id>534-22-5252</id>
 <name>Bob Citizen</name>
 <language>Spanish</language>
 <rating>3.2</rating>
</student>
```

How could a program distinguish the elements with the same names but different meanings?

Namespaces in XML



Namespaces in XML



A namespace is a set of names in which all names are unique.

The name 'title' can now be identified as: Book.title, Project.title, Employee.title ...

These names are called "qualified names".

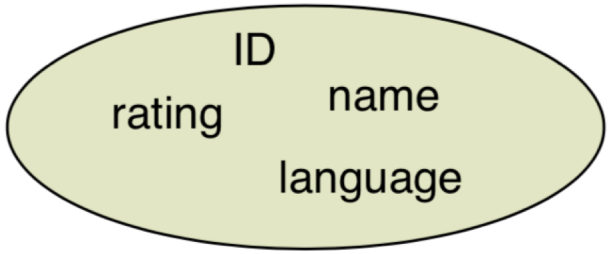
XML namespaces give elements and attributes a unique name across the Internet.

XML namespaces enable programmers to process the tags and attributes they care about and ignore those that don't matter to them.

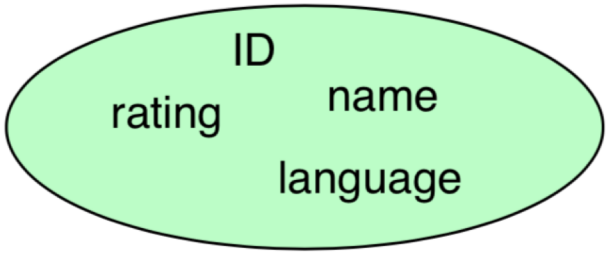
Namespaces in XML

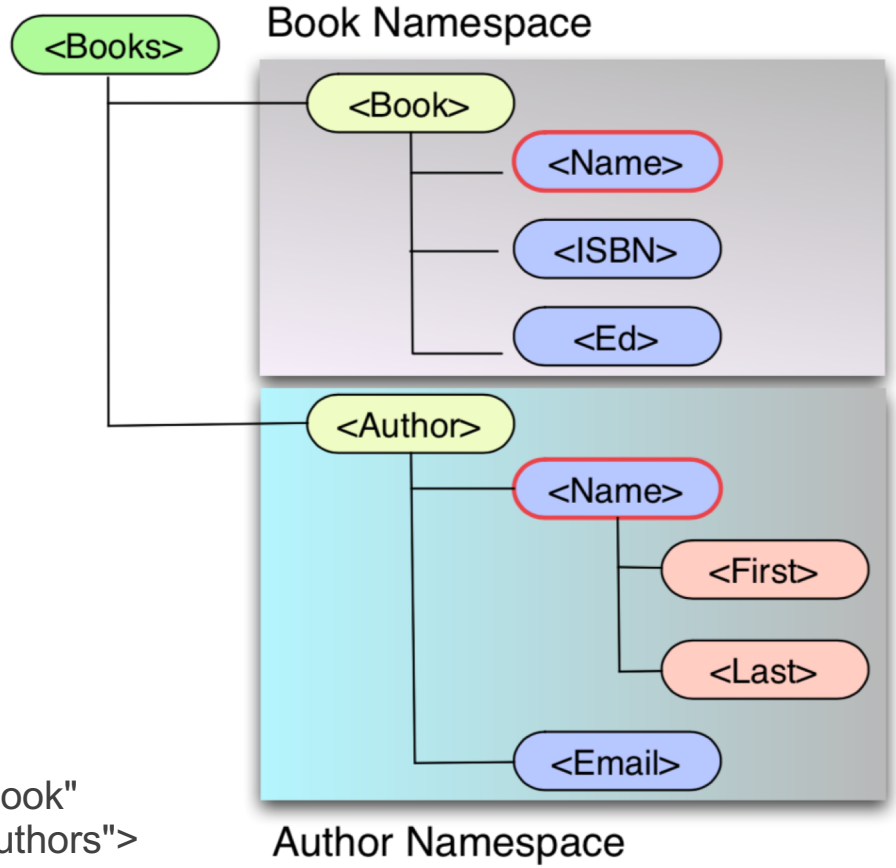
Elements with namespaces: programmatically distinguishable now …

namespace:UniversityX/Student



namespace:UniversityY/Student



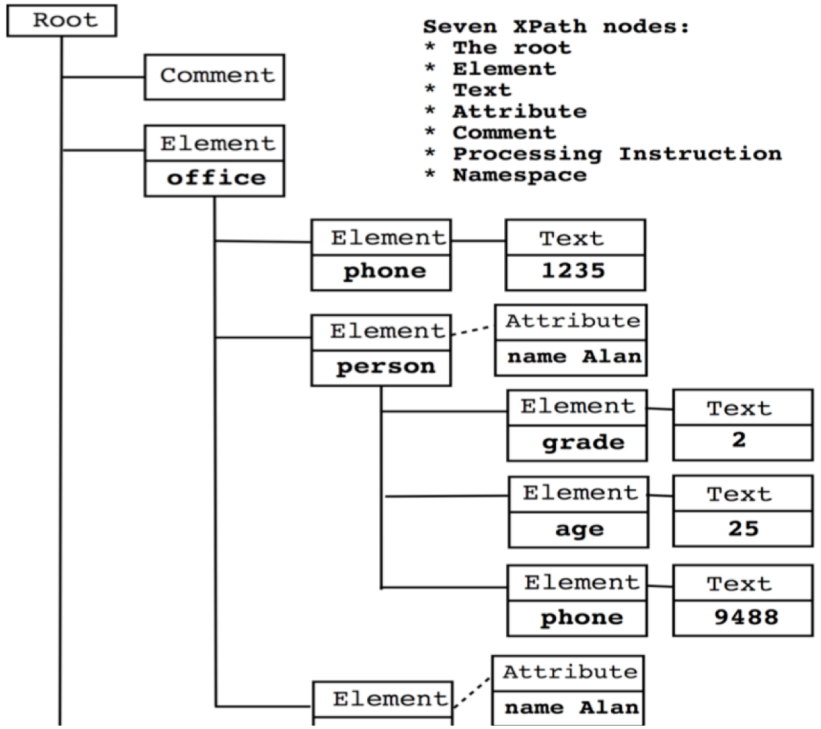



```
<Books
    xmlns:book="http://booknamespace/book"
    xmlns:author=”http://mynamespace/authors">
```

Accessing/Querying XML files

XPath is a small, “embedded” query language for XML documents.

```
<!-- Some simple test data -->
<office>
  <phone>1235</phone>
  <person name="Alan">
    <grade>2</grade>
    <age>25</age>
    <phone>9488</phone>
  </person>
  <person name="Sue">
    <grade>8</grade>
    <age>29</age>
    <phone>2044</phone>
  </person>
  <person name="Vicky">
    <grade>2</grade>
    <age>29</age>
    <phone>2738</phone>
  </person>
  <person name="Mike">
    <grade>3</grade>
    <age>25</age>
    <phone>6995</phone>
  </person>
</office>
<!-- four lonely people -->
```



Example 1: Identify all person elements.

Example 2: Identify all person elements with grade 2.

Example 3: Identify Sue’s age.

Accessing/Querying XML files

XQuery is a declarative language in which a query is represented as an expression

Sample XQuery

```
<bib>
{
  for $b in doc("http://bstore1.example.com/bib.xml")/bib/book
  where $b/publisher = "Addison-Wesley" and $b/@year > 1991
  return
    <book year="{ $b/@year }">
     { $b/title }
     { $b/author}
    </book>
}
</bib>
```

Can you "read" it?

file: select-v2.xq

Accessing/Querying XML files

```
for $book in doc("books.xml")//BOOKS/ITEM,
    $quote in doc("quotes.xml")//listing/ITEM
where $book/ISBN = $quote/ISBN
return
 <book>
   {$book/TITLE, $quote/PRICE}
 </book>
```

Join two documents

```
<results> {
for $b in doc("bib.xml")//book
return
<holding>
  {
    $b/title,
    if ($b/@type = "Journal")
       then $b/editor
       else $b/author
  }
</holding>
}
</results>
```

If/else conditions

Advantages/Disadvantages of NoSQL

Cheap, easy to implement

Data are replicated and can be partitioned

Easy to distribute

Don't require a schema

Can scale up and down

Quickly process large amounts of data

Relax the data consistency requirement (CAP)

New and sometimes buggy

Data is generally duplicated, potential for inconsistency

No standardized schema

No standard format for queries

No standard language

Difficult to impose complicated structures

Depend on the application layer to enforce data integrity

Too many options, which one, or ones to pick ?

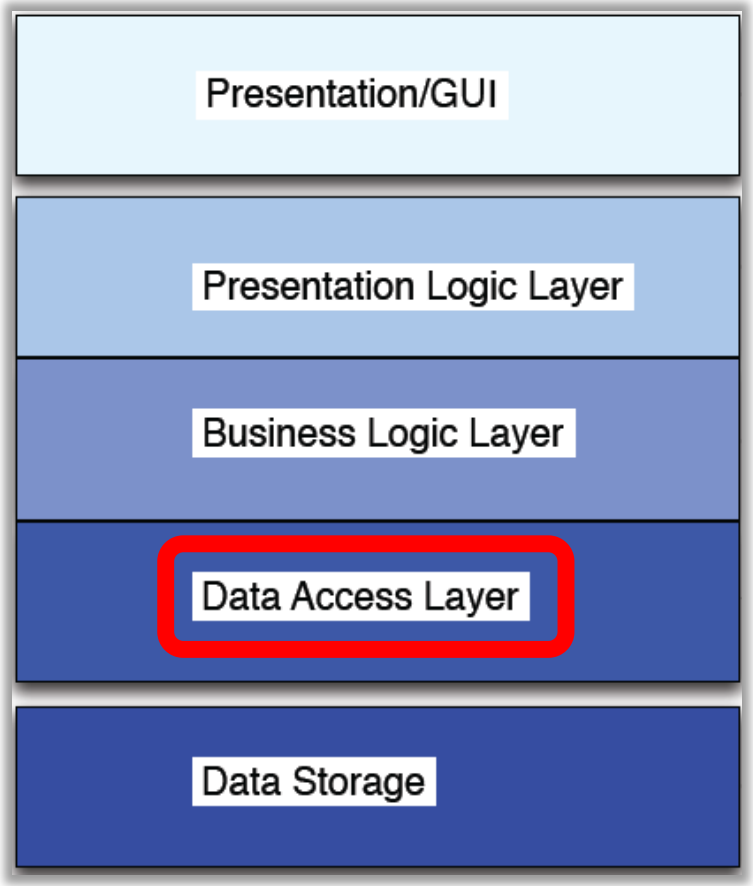
Accessing DB from an application …

When you work with a database system (regardless of its storage model) in an application, the code issues a query statements to the database via some form of “data-connectivity API”

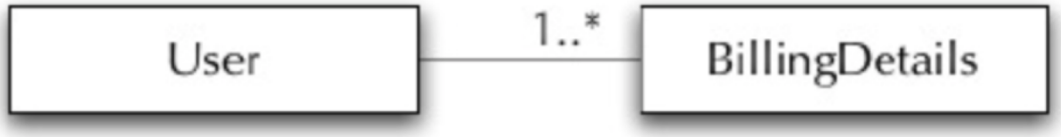
The application code blocks relating to using this library form “Data Access Layer” in the stack.

- For objects to persists, we need to convert the object values into the values that can be stored in the storage (relational tables, this case) – and convert them back upon retrieval.

- This should be done while preserving the properties of the objects and their relationships



Impedance (or Paradigm) Mismatch Problem




```
public class User  {
   private String userName;
   private String name;
   private String address;
   private Set billingDetails;
   // accessor methods
}
```

```
public class BillingDetails  {
   private String accountNumber;
   private String accountName;
   private String accountType;
   private User user;
   // accessor methods
}
```

```
create table User (
   username varchar(15)
         not null primary key,
   name varchar(50) not null,
   address varchar(100)
)
```

```
create table Billing_Details (
   account_number varchar(10)
                  not null primary key,
   account_name varchar(50) not null,
   account_type varchar(2) not null,
   username varchar(15)
         foreign key references user
)
```

Impedance Mismatch Problem

Granularity Problem

We want to create a separate `Address` class as other classes in the system may also carry address information.

How should this be represented in relational tables?

- Should we add an Address table?
- Should we add an Address column to the User table instead?
- Should the Address be a string? Or multi-columns?
  - ***Coarse Granularity***, as a single field
  - ***Fine Granularity,*** as multiple fields

address = 200 2nd Ave. South #358, St. Petersburg, FL 33701-4313 USA

street address = 200 2nd Ave. South #358
city = St. Petersburg
postal code = FL 33701-4313
country = USA

Impedance Mismatch Problem

In application code:

Identity Concept Mismatch

Objects can be either equal or identical:

identical = same object (address)

equal = same values

In RDB, these two separate concepts do not exists. There is only one concept of identify = primary key. (i.e., same primary key -> same objects)

Potentially problematic, if duplicate objects are considered the same object (or vice versa) in database

Impedance (or Paradigm) Mismatch Problem

Association Problem

- User, Address and BillingDetails classes are associated (different kind of associations - represented differently in tables)
- Association mapping and the management of the entity associations are central concept of any object persistence solution.
- OO languages represent associations using *object references* and collections of object references

Object references are directional; the association is from one object to another. To be able to navigate 'between' objects, one needs to define the association *twice*.

```
public class User {
  private Set billingDetails; ...
}
```

```
public class BillingDetails {
  private User user; ...
}
```

Impedance Mismatch Problem

Object Graph Navigation

In OO, method chaining like:

User.getBillingDetails().getAccountNumber() is commonly done …

From a user, you access the billing information, from that, you access the account number …

However, this is not an efficient way to retrieve data from relational tables (i.e., instead of accessing single objects, you'd do joins …)

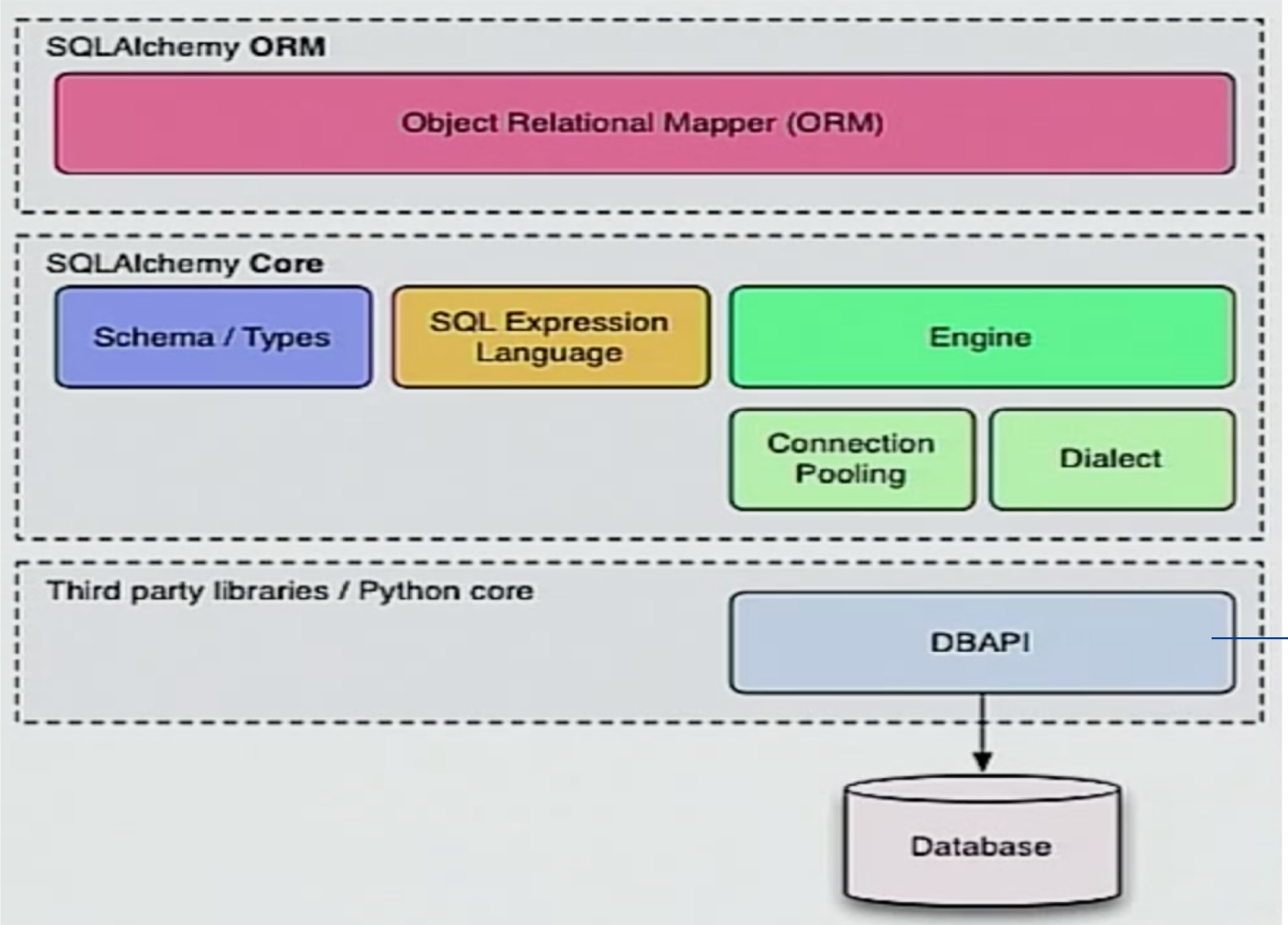
```
select * from USER u where u.USER_ID = 123
```

if we need to retrieve the same User and then subsequently visit each of the associated BillingDetails instances, we use a different query:

```
select *
from USER u, BILLING_DETAILS bd
where u.USER_ID = bd.USER_ID
and u.USER_ID = 123
```

Object Relational Mapping (ORM)

e.g., Python SQLAlchemy



(Optional)

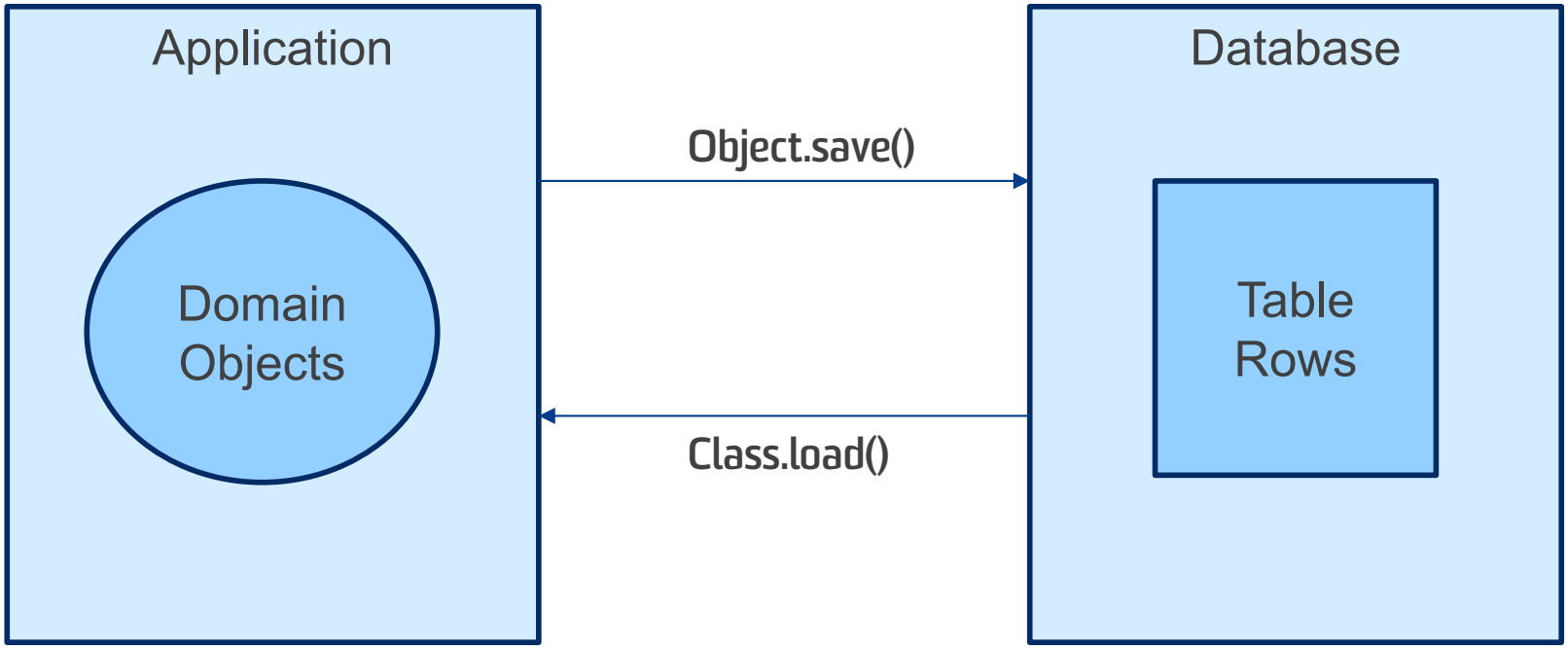
SQL access management libraries"

Python standard/core Low-level database communication layer (many implementations)

- psycopg2 (PostgreSQL)
- cx_oracle (Oracle)
- pyodbc (ODBC)

ORM

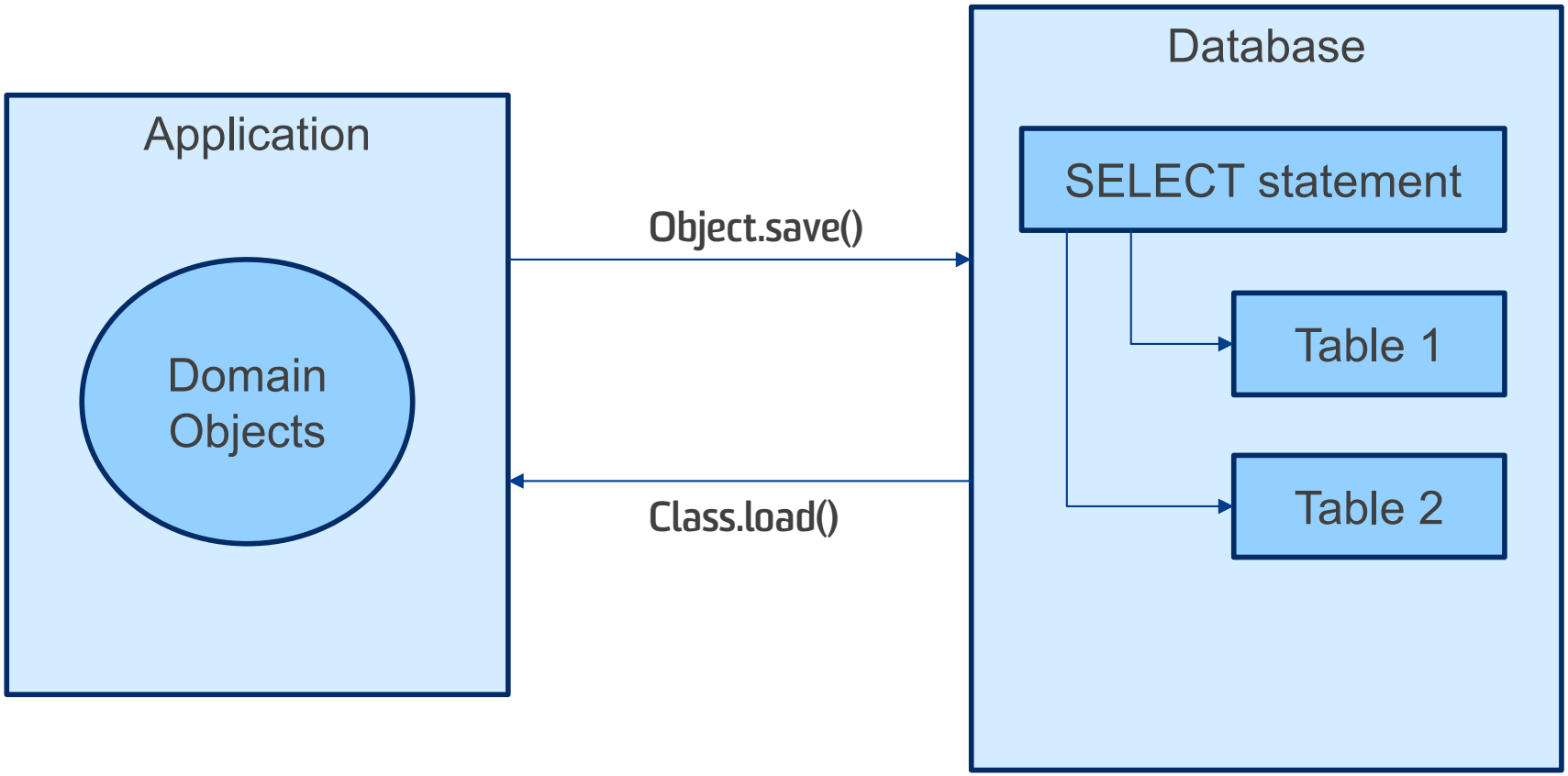
ORM is the process of associating object oriented classes (your application domain model) with database tables



http://techspot.zzzeek.org/category/sqlalchemy/

ORM

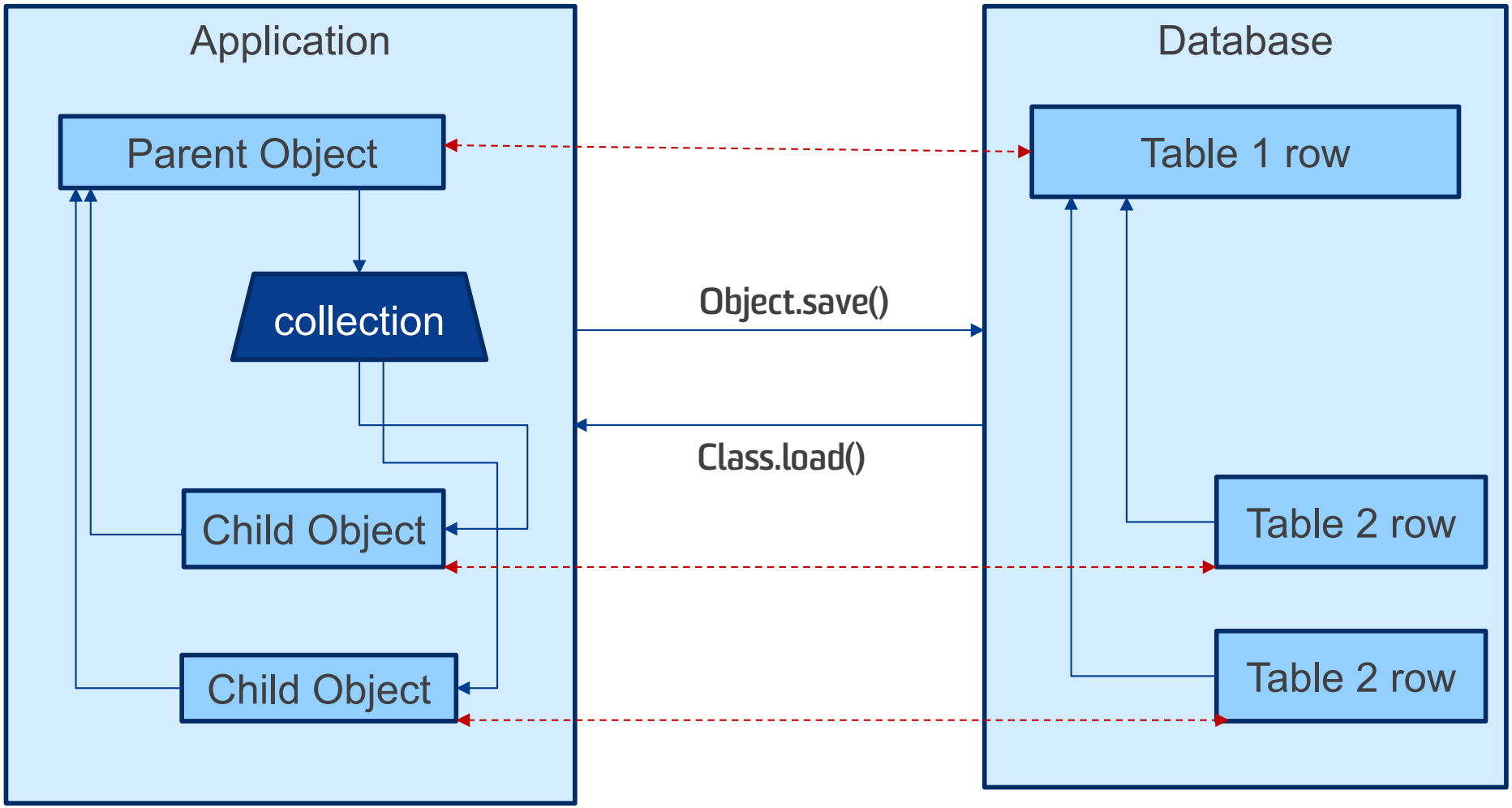
Some ORM can represent arbitrary rows as domain objects – e.g., rows derived from SELECT statement joining multiple tables



http://techspot.zzzeek.org/category/sqlalchemy/

ORM

Most ORM represent basic ‘compositions’, 1-M, M-1 using foreign key associations



http://techspot.zzzeek.org/category/sqlalchemy/

Flavours of ORM

Two general “styles” of ORM – Data Mappers and Active Record.

Active Record has domain object handle their own persistence.

```
user_record = User(name="ed", fullname="Ed Jones")
user_record.save()

user_record = User.query(name='ed').fetch()
user_record.fullname = "Edward Jones"
user_record.save()
```

Object.save() => the persistence logic is “attached” to the object.

Save() == Insert/Update

Flavours of ORM

Two general “styles” of ORM – Data Mappers and Active Record.

Data Mapper tries to separate the details of persistence from the objects themselves

```
dbsession = start_session()

user_record = User(name="ed", fullname="Ed Jones")

dbsession.add(user_record)

user_record = dbsession.query(User).filter(name='ed'
user_record.fullname = "Edward Jones"

dbsession.commit()
```

The idea of ‘session’

Objects and their persistence API are associated with the session (i.e., the objects you work with themselves does not have persistence API)

Flavours of ORM

They also come in two different ‘styles’ of configuration patterns.

Most use ‘all-in-one’, or declarative style where class and table information is together

```
# a hypothetical declarative system

class User(ORMObject):
    tablename = 'user'

    name = String(length=50)
    fullname = String(length=100)

class Address(ORMObject):
    tablename = 'address'

    email_address = String(length=100)
    user = many_to_one("User")
```


http://techspot.zzzeek.org/category/sqlalchemy/

Flavours of ORM

They also come in two different ‘styles’ of configuration patterns.

Less common style – keeping the class and table schema information separate

```
# class is declared without any awareness of databas
class User(object):
    def __init__(self, name, username):
        self.name = name
        self.username = username


# elsewhere, it's associated with a database table
mapper(
    User,
    Table("user", metadata,
          Column("name", String(50)),
          Column("fullname", String(100))
    )
)
```

Your model stays ‘pure’ – unaware of its persistence structure, but the configuration task becomes repetitive …

SQLAlchemy ORM

It is a Data Mapper style ORM, with declarative style configuration

ORM builds on SQLAlchemy Core

In contrast to the SQL Expression Language which presents schema-centric view of the data, ORM provides domain-model centric view of the data

```
>>> from sqlalchemy import Column, Integer, String

>>> class User(Base):
...     __tablename__ = 'user'

...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     fullname = Column(String)

...     def __repr__(self):
...         return "<User(%r, %r)>" % (
...                 self.name, self.fullname
...             )

>>>
```

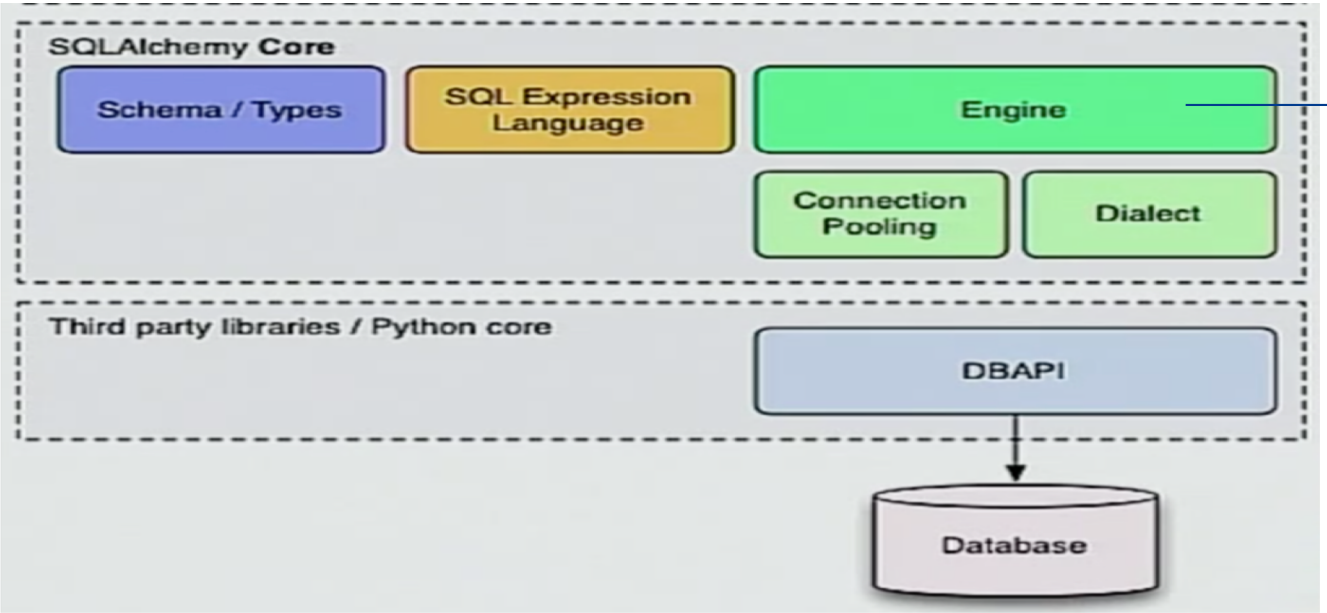
After this, User class now has an associated Table called 'user'

Directly Executing SQL (SQLAlchemy Core)

SQLAlchemy

- "Uniform" SQL access to relational databases (in SQLAlchemy way)
- i.e., SQL access library built on top of the DBAPI connectivity

```
from sqlalchemy import create_engine
engine = create_engine('postgresql://usr:pass@localhost:5432/sqlalchemy')
…
engine = create_engine('sqlite:///some.db')
```



Object that maintains
The classical DBAPI interactions
With the database

Directly Executing SQL (SQLAlchemy Core)

```
users = Table('users', metadata,
   Column('id', Integer, Sequence('user_id_seq'), primary_key=True),
   Column('name', String(50)),
   Column('fullname', String(50)),
   Column('password', String(12))
)
```

SQL Expression Language

```
>>> ins = users.insert()
>>> conn.execute(ins, id=2, name='wendy', fullname='Wendy Williams')
```

```
INSERT INTO users (id, name, fullname) VALUES (?, ?, ?)
(2, 'wendy', 'Wendy Williams')
COMMIT
```

```
>>> from sqlalchemy.sql import select
>>> s = select([users])
>>> result = conn.execute(s)
```

```
SELECT users.id, users.name, users.fullname
FROM users
()
```

SQLAlchemy ORM

Use ‘declarative’ style configuration – User class with table configuration …

```
sqlalchemy_tutorial — python .venv/bin/sliderepl 04_orm_basic.py — 70×32

+-------------------------------------------------------------------+
| *** Object Relational Mapping ***                                 |
+-------------------------------------------------------------------+
| The *declarative* system is normally used to configure            |
| object relational mappings.                                       |
+------------------------------------------------- (1 / 42) --+

>>> from sqlalchemy.ext.declarative import declarative_base
>>> Base = declarative_base()

>>>

+-------------------------------------------------------------------+
| a basic mapping.  __repr__() is optional.                         |
+------------------------------------------------- (2 / 42) --+

>>> from sqlalchemy import Column, Integer, String

>>> class User(Base):
...     __tablename__ = 'user'

...     id = Column(Integer, primary_key=True)
...     name = Column(String)
...     fullname = Column(String)

...     def __repr__(self):
...         return "<User(%r, %r)>" % (
...                 self.name, self.fullname
...             )

>>>
```

SQLAlchemy ORM

Instead of the ‘Engine’, application developers will deal with ‘Sessions’

```
>>> from sqlalchemy import create_engine
>>> engine = create_engine('sqlite://')
>>> Base.metadata.create_all(engine)
```

```
>>> from sqlalchemy.orm import Session
>>> session = Session(bind=engine)
```

```
>>> session.add(ed_user)
```

```
>>> our_user = session.query(User).filter_by(name='ed').first()
```

We can play the Mike Bayer’s slides some more here to look at the features …

Managing and Publishing Metadata

Metadata, what is it?

Managing and Publishing Metadata

Nearly every device we use relies on metadata or generates it …

Edward Snowden – a contractor at United States National Security Agency exposed how the agency collected metadata on telephone calls directly from telecommunications companies (note: only metadata, not the actual conversations)

But how much information could be inferred about individuals from only metadata ?

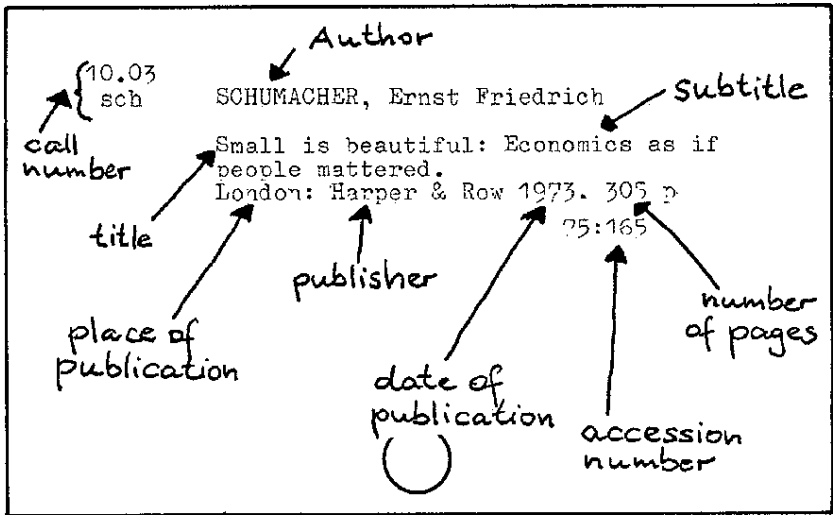
Possible metadata collected about a phone call:

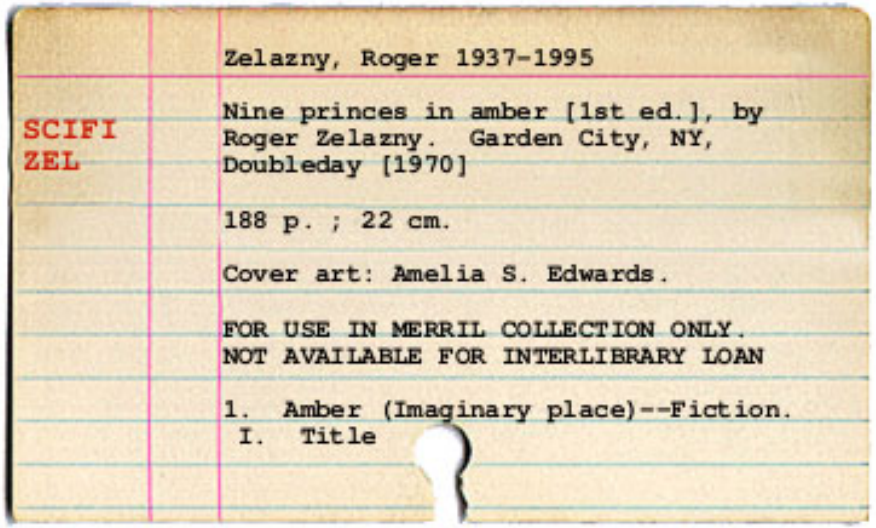
- Phone numbers (caller, recipient)
- Time and duration of the call
- Mobile phone locations (caller, recipient)
  - If mobile phone is in connection with local cell towers, a record of your location at any given moment …

Metadata is becoming as important as the data itself. Naturally, data services APIs should be aware of metadata and know how to publish and consume metadata along with the data.

Managing and Publishing Metadata

Librarians have been working with metadata for centuries …





In the end, they are data – which can be modelled, stored and managed now …

Title	Author	Date of publication	Subject	Call number	Pages
Intellectual Pro-	Palfrey,	2012	Intellectual proper-	HD53 .P35	172

Managing and Publishing Metadata

Why do we need metadata when we have data object itself?

- Metadata is a “map”, is a means by which the complexity of an object is represented in a simpler form
  - A roomful of books is not called a library, books + catalog is. The catalog provides a simplified representation of the materials in the library collection.
  - Primarily, metadata helps with ‘resource discovery’ – the process by which information resources that might be relevant to your need is identified.

Descriptive metadata: description of an object

Administrative metadata: information about the origin and maintenance of an object

e.g., a photograph digitized using a specific type of scanner at a particular resolution, with some restrictions on copyright, etc.

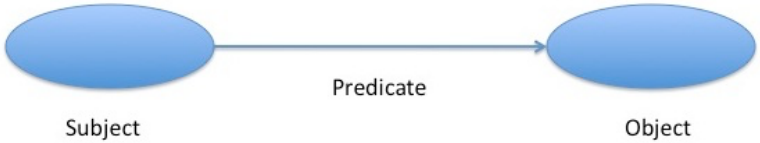
Structural metadata: information about how an object is organised (e.g., ToC)

Provenance metadata: traces/processes involved in producing the object.

Describing Description …

Metadata is a statement about a potentially informative “thing” (resource).

A well adopted metadata description language is RDF (resource description framework)

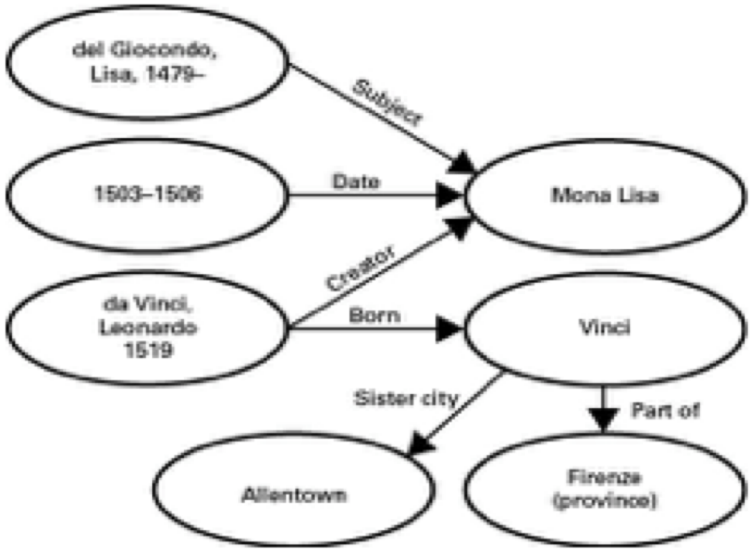


Subject: Mona Lisa Predicate: Creator Object: Da Vinci

Subject refers to the ‘entity’ being described

Object refers to another entity being used to describe the subject …

RDF Triples – could be a useful data model just by itself … (graphs -> network analysis -> gets interesting …!!)



https://www.w3.org/TR/rdf11-primer/

Descriptive Metadata

Element	Definition
Contributor	An entity responsible for making contributions to the resource.
Coverage	The spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the resource is relevant.
Creator	An entity primarily responsible for making the resource.
Date	A point or period of time associated with an event in the lifecycle of the resource.
Description	An account of the resource.
Format	The file format, physical medium, or dimensions of the resource.
Identifier	An unambiguous reference to the resource within a given context.
Language	A language of the resource.
Publisher	An entity responsible for making the resource available.
Relation	A related resource.
Rights	Information about rights held in and over the resource.
Source	A related resource from which the described resource is derived.
Subject	The topic of the resource.
Title	A name given to the resource.
Type	The nature or genre of the resource.

Standard, the simplest form of metadata: **Dublin Core**

Originally developed to help improve the search engine and indexing the web documents

Title: Mona Lisa
Title: La Gioconda
Creator: Leonardo da Vinci
Subject: Lisa Gherardini
Date: 1503–1506

RDF Example (with Dublin Core)

https://www.w3schools.com/xml/xml_rdf.asp

How would you add metadata do your assignment 2?
What would be useful descriptions for your data?