



**UNSW**  
SYDNEY

Australia's  
Global  
University

**COMP9321**

# **Data Services Engineering**

**Term 1, 2019**

**Week 1 Lecture 2**

# We are Generating Vast Amount of Data

since this slides. we have aenerated....

## Insert Web Page

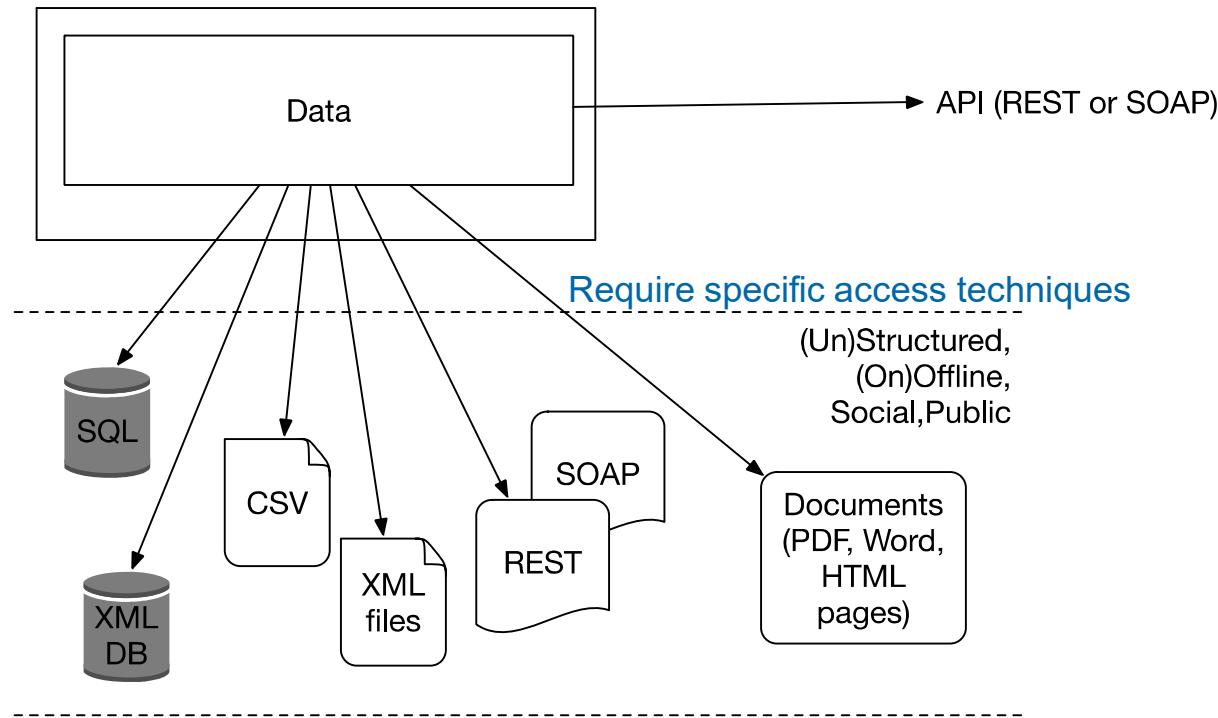
This app allows you to insert secure web pages starting with https:// into the slide deck. Non-secure web pages are not supported for security reasons.

Please enter the URL below.

https:// www.webfx.com/internet-real-time/

Note: Many popular websites allow secure access. Please click on the preview button to ensure the web page is accessible.

# Data Services – what is it about?



- Data integration /aggregation from multiple sources (preparation)
- Data publication for consumer access (API/ Visualization)
- Some data processing or analytics

# Challenges from implementation view point

Difficult to obtain a "single view of X" for any X

- » What data do I have about X?
- » How do I stitch together the info I need (choose the right data model)?
- » What else is X related to?

No uniformity (model or language)

- » Data about X is stored in many different formats.
- » Accessing or updating X involves many different APIs or access methods
- » Manual coding of "distributed query plans"

What's data sources or existing APIs available and where?

What protocol do they use?

What format are they in?

# Obtaining Data

Useful data can be found in many places

- » on the Web, possibly via an API/APP/Mobile...
- » in documents in a file system
- » in spreadsheets
- » in videos
- » ... name a few

and in a variety of formats

- » Unformatted text (in files)
- » PDF documents (in files)
- » HTML documents (web pages)
- » XML documents (via web APIs)
- » JSON data (often via web APIs)
- » CSV data files (spreadsheets)

# Unformatted Text Data

Unformatted text is generally *unfriendly*

Hi James,

Here are some stats on in the **first term of 2018**.

**Up by 20%** on last year for our **end point solutions**  
**(backup solutions** sales ranking **highest**).  
And **corporate solutions** sales is **up by 30% !!**  
**Vulnerability scanning** solutions ranking **highest**.

Best regards,

Bob

# Unformatted Text Data (Cont'd)

The same information can be conveyed in many ways

Hey James,

FYI

Sales for the **end point solutions** increased by **20%** where **backup solutions** being most popular in **first term 2018**.

And **corporate solutions'** sales had a great jump by **30%** with an increase demand for vulnerability scanners.

Regards,

Bob

# Unformatted Text Data (Cont'd)

In order to mine useful data from text, sophisticated techniques are required like

- » Natural language processing (NLP) (syntax)
- » Other Machine Learning (ML) (patterns recognition etc.)

Such techniques are, at best, approximate

and usually are for specific problems with some pre-defined knowledge

- » Named entity recognition (who/what is end point solutions?)
- » Training requirements (for ML)

# Unformatted IoT Data (SmartHome)

|            |                 |      |     |               |       |
|------------|-----------------|------|-----|---------------|-------|
| 2010-11-04 | 05:40:51.303739 | M004 | ON  | Bed_to_Toilet | begin |
| 2010-11-04 | 05:40:52.342105 | M005 | OFF |               |       |
| 2010-11-04 | 05:40:57.176409 | M007 | OFF |               |       |
| 2010-11-04 | 05:40:57.941486 | M004 | OFF |               |       |
| 2010-11-04 | 05:43:24.021475 | M004 | ON  |               |       |
| 2010-11-04 | 05:43:26.273181 | M004 | OFF |               |       |
| 2010-11-04 | 05:43:26.345503 | M007 | ON  |               |       |
| 2010-11-04 | 05:43:26.793102 | M004 | ON  |               |       |
| 2010-11-04 | 05:43:27.195347 | M007 | OFF |               |       |
| 2010-11-04 | 05:43:27.787437 | M007 | ON  |               |       |
| 2010-11-04 | 05:43:29.711796 | M005 | ON  |               |       |
| 2010-11-04 | 05:43:30.279021 | M004 | OFF | Bed_to_Toilet | end   |
| 2010-11-04 | 05:43:45.7324   | M003 | ON  | sleeping      | begin |
| 2010-11-04 | 05:43:52.044085 | M003 | OFF |               |       |
| 2010-11-04 | 05:43:53.185335 | M002 | ON  |               |       |
| 2010-11-04 | 05:43:53.253809 | M003 | ON  |               |       |
| 2010-11-04 | 05:43:59.493281 | M002 | OFF |               |       |
| 2010-11-04 | 05:44:04.048766 | M003 | OFF |               |       |
| 2010-11-04 | 05:44:06.14204  | M003 | ON  |               |       |
| 2010-11-04 | 05:44:11.229146 | M003 | OFF |               |       |

# PDF Documents

PDF documents have some structure

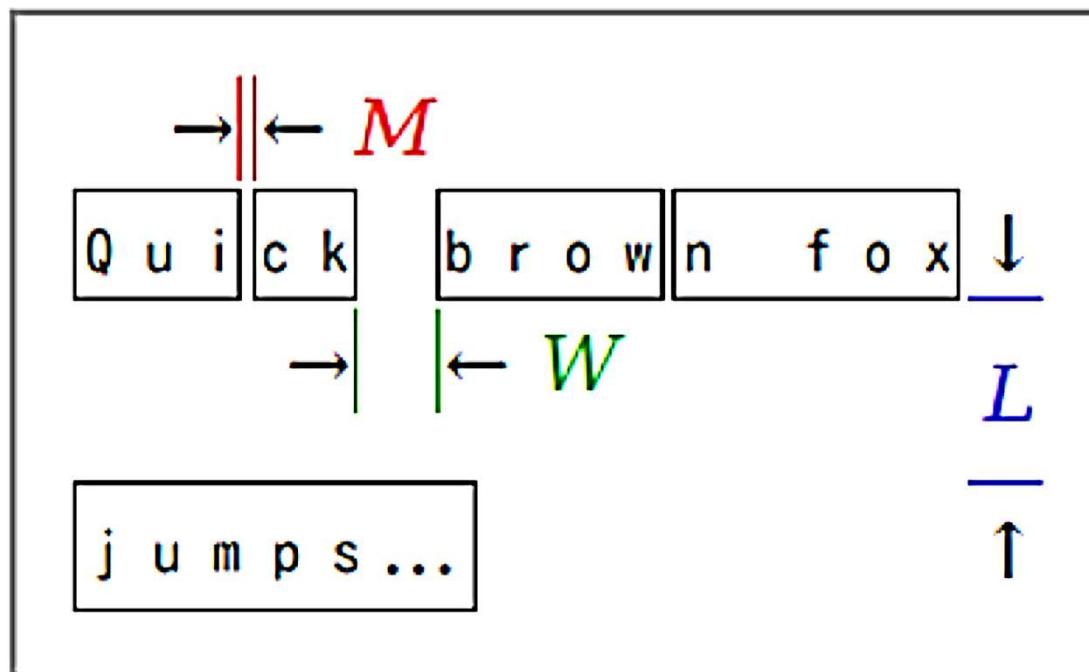
- » which specifies content and layout commands
- » including both text and (mainly) binary data
- » but structure is not necessarily helpful for extraction

Python packages

- e.g. pyPDF2
  - » aim to extract the text from the document
  - » but don't aim to keep the layout structure
- e.g. pyOCR
  - » OCR based approach
  - » may keep the layout
  - » may not be so accurate

# PDF Documents (Cont'd)

Why even extracting text from PDFs is hard ...  
text chunks don't necessarily correspond to words



# PDF Documents (Cont'd)

More sophisticated extraction from PDF

PDFminer (including pdf2txt.py and dumppdf.py)

- » toolset implemented in Python (v2)
- » parses PDF documents, both text & images
- » can convert PDFs to other formats (e.g. HTML)
- » can output text + layout information

But still requires significant work to extract *data*

- » requires NLP/ML, but aided by layout information

# PDF Documents (Cont'd)

Tables are often sources of useful data, but require ...

finding table boundaries

finding rows and columns

finding cell boundaries

extracting text from cells

etc. etc.

Above is easy in HTML

Much harder in PDF

But e.g. [pdftables.com](http://pdftables.com)

| State      | City         | Town      | POP |
|------------|--------------|-----------|-----|
| New York   | Rensselaer   | Troy      | 1   |
|            |              | Brunswick | 2   |
|            | St. Lawrence | Potsdam   | 3   |
|            |              | Canton    | 4   |
| California | San Diego    | Coronado  | 5   |
|            |              | Del Mar   | 6   |
|            |              | Malibu    | 7   |
|            | Los Angeles  | Compton   | 8   |

# HTML Documents

HTML documents include explicit markup

- » which is more *semantic* than PDF layout data
- » making it easier to recognise document components
- » but content is still semi-structured (“at creator’s whim”)

However, much HTML these days is generated

- » giving a regular structure which is parseable

# HTML Documents (Cont'd)

Example HTML (1)

```
<h1>Sales</h1>
<ul>
<li> Anti-virus solution 50000 </li>
<li> Backup solutions 150000 </li>
</ul>
```

Easy to find structure; recognising columns harder.

# HTML Documents (Cont'd)

## Example HTML (2)

```
<h1>Sales</h1>
<table>
<tr> <th>Solution</th> <th>Quantity</th> </tr>
<tr> <td>Anti-virus</td> <td>50000</td> </tr>
<tr> <td>Backup</td> <td>150000</td> </tr>
</table>
```

Easy to find columns; headings assist with semantics.

# HTML Documents (Cont'd)

The Python BeautifulSoup library allows analysis of HTML

Example:

```
<!doctype html>
<html>
    <head><title> A simple page </title></head>
    <body>
        <p> Some simple content.</p>
    </body>
</html>
```

Assume above is in <http://sec.com/p.html>

# HTML Documents (Cont'd)

BeautifulSoup Example (cont):

```
import requests
from bs import BeautifulSoup

page = requests.get("http://sec.com/p.html")

# produce a nested structure representing the HTML
soup = BeautifulSoup(page.content, 'html.parser')

# get subcomponents of the html component
html = list(soup.children)[2]

# get the <body>...</body>
body = list(html.children)[3]

# get the <p>...</p> text
p = list(body.children)[1]
para = p.get_text()
```

# HTML Documents (Cont'd)

Combined with other Python modules e.g. regexps

BeautifulSoup provides powerful tools to extract text

can also place text in context within page structure

can allow extraction of structured data

- » from within known page structures
- » or using specific patterns of tags/text
- » which is often the case nowadays
- » when HTML is mostly generated by scripts

# HTML Documents (Cont'd)

Example: UNSW Handbook

Easy to navigate and scrape because ...

- » all program/stream/course pages have same structure
- » significant amount of cross-linkage among pages
- » index pages give links to all pages of given type

<http://www.handbook.unsw.edu.au/2018/index.html>

UNSW SYDNEY Handbook 2018

YOUR STUDY PLANNERS  
ARE HERE.  
HEAD TO: STUDENT3PLUS.UNSW.EDU.AU

MyUNSW Current Students Future Students Class Timetable eLearning Library Contacts

View Previous Handbook Editions

Current Handbook

ARCHIVE EDITION

**Handbook 2018**

your guide to degree programs and courses offered at UNSW

**UNSW Study Choices**

**Undergraduate Study**

For:

- future UG students
- current UG students

**Postgraduate Coursework Study**

For students who:

- have completed a UG degree and are interested in PG coursework mode

**Postgraduate Research Study**

For students who:

- have completed a UG degree and are interested in PG research mode

**Non-Award Study**

For students who:

- wish to study without award of a formal degree, diploma or certificate

**SEARCH THE UNSW HANDBOOK**

**Quick Search**

Select a Study Level and enter a word or phrase for the **Program**, **Stream**, **Course** or **Specialisation** you're looking for.

Study Level

Word / Phrase

**UNSW Quick Links**

Domestic Undergraduate Applicants Domestic Postgraduate Applicants RESEARCH APPLICANTS International Applicants ACADEMIC CALENDAR

moodle Lecture Recordings APPLY ONLINE FEES FEE ESTIMATOR

URL for this page: <http://legacy.handbook.unsw.edu.au/2018/index.html>

© UNSW Sydney (CRICOS Provider No. 00098G), 2004-2017. The information contained in this Handbook is indicative only. While every effort is made to keep this information up-to-date, the University reserves the right to discontinue or vary arrangements, programs and courses at any time without notice and at its discretion. While the University will try to avoid or minimise any inconvenience, changes may also be made to programs, courses and staff after enrolment. The University may also set limits on the number of students in a course.

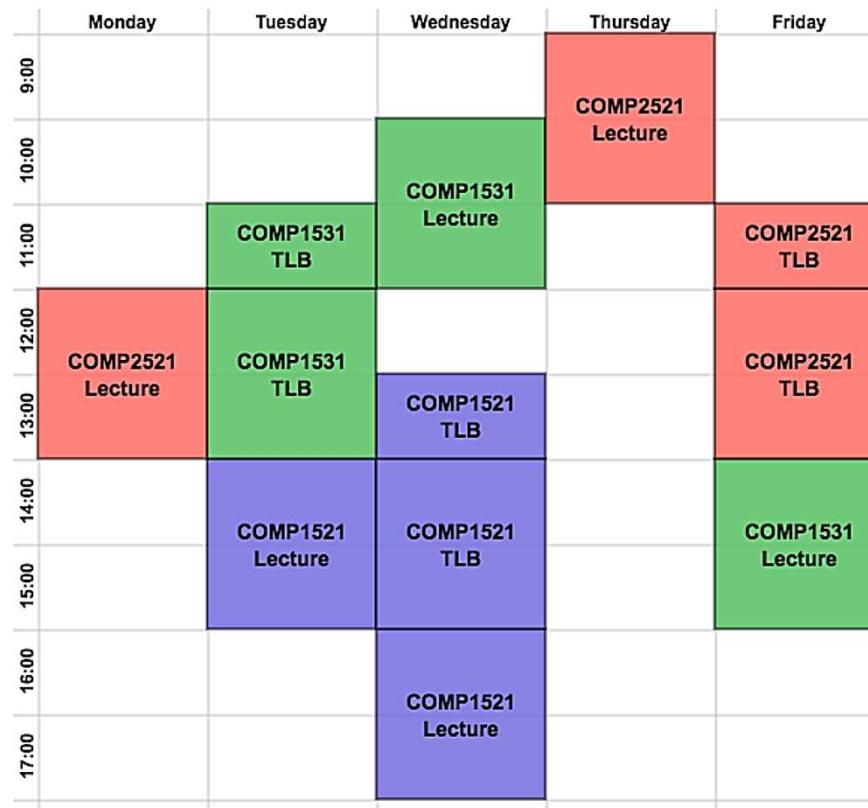
Authorised by Deputy Vice-Chancellor (Education)  
CRICOS Provider Code 00098G | ABN: 57 195 873 179

Privacy Policy | Copyright and Disclaimer | Accessibility | Site Feedback



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
3 <html xmlns="http://www.w3.org/1999/xhtml"
4   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
5   xml:lang="en">
6   <head>
7     <meta name="generator" content="UNSW - Handbook 2018" />
8     <meta http-equiv="content-type" content="text/html; charset=utf-8" />
9     <meta http-equiv="content-script-type" content="text/javascript; charset=utf-8" />
10    <title>UNSW - Handbook 2018</title>
11
12    <link rel="stylesheet" href="/styles/2018/screen.css" type="text/css" media="screen" charset="utf-8" />
13    <link rel="stylesheet" href="/styles/2018/colour.css" type="text/css" media="screen" charset="utf-8" />
14
15    <script src="/scripts/jquery-1.6.js" type="text/javascript" charset="utf-8"></script>
16    <script src="/scripts/hbJavascript/hbStandard.js" type="text/javascript" charset="utf-8"></script>
17    <script src="/scripts/bValidation.js" type="text/javascript" charset="utf-8"></script>
18  </head>
19  <body id="home">
20
21    <!-- Page Wrapper -->
22    <div id="pageWrapper">
23      <!-- Header Wrapper -->
24      <div id="headerWrapper">
25        <div id="header">
26          <div id="unsw">
27            <a href="http://www.unsw.edu.au" onclick="window.open(this.href); return false;" title="The University of New South Wales"></a>
28            <h1>Handbook 2018</h1>
29          </div>
30          <div id="your-study-planner">
31            <a href="http://www.student3plus.unsw.edu.au" onclick="window.open(this.href); return false;" title="Your Study Planners"></a>
32          </div>
33        </div>
34      </!-- End of Header Wrapper -->
35
36
37      <!-- Content Wrapper -->
38      <div id="contentWrapper">
39
40        <!-- Content -->
41        <div id="content">
42          <!-- Top Navigation Bar -->
43          <div id="navigationBar">
44            <div class="left">
45              <ul id="actions">
46                <li><a href="https://my.unsw.edu.au/">MyUNSW</a></li>
47                <li><a href="https://student.unsw.edu.au/">Current Students</a></li>
48                <li><a href="http://www.futurestudents.unsw.edu.au/">Future Students</a></li>
49                <li><a href="http://timetable.unsw.edu.au/">Class Timetables</a></li>
50                <li><a href="https://student.unsw.edu.au/node/62">eLearning</a></li>
51                <li><a href="http://www.library.unsw.edu.au/">Library</a></li>
52                <li><a href="#">Contact</a></li>
53              </ul>
54            </div>
55          </div>
56        </!-- Content -->
57      </div>
58
59      <!-- Footer -->
60      <div id="footer">
61        <div>
62          
63          <div>
64            
65            
66            
67            
68            
69            
70          </div>
71        </div>
72      </div>
73
74    </!-- Content Wrapper -->
75  </body>
76
77
```

# HTML Documents (Cont'd)



Example: Bojangles Timetable generator implemented by CSE student using data scraped from classutil

Source: [tdransfield.net/utilities/bojangles](http://tdransfield.net/utilities/bojangles)

# XML and JSON Data

XML and JSON are already structured data

- » good for representing hierarchical structure
- » have tags to indicate type of data (metadata)
- » have much software to traverse their content

Tags help to massage into target structure

Extensible Markup Language (XML) is a set of rules established by the W3C organization. The rules provide a framework for uniformly encoding documents in a human readable form.

XML is similar to HTML but all the are user-defined.

# Why XML

The main role of XML is to facilitate a transparent exchange of data over the Internet.

- Example of technologies using XML include: RSS , Atom, SOAP, XHTML, KML, Xquery, Xpath, OpenDocument, OpenMath, etc.(see [http://en.wikipedia.org/wiki/List\\_of\\_XML\\_markup\\_languages](http://en.wikipedia.org/wiki/List_of_XML_markup_languages))
- Several document management productivity tools default to XML format for internal data storage. Example: Microsoft Office, OpenOffice.org, and Apple's iWork.
- Android OS relies heavily on XML to save its various resources such as layouts, string-sets, manifest, etc.

# XML and JSON Data (Cont'd)

Example XML

```
<?xml version="1.0"?>
<countries>
    <country name="Liechtenstein"> ←
        <rank>2</rank> <year>2008</year>
        <neighbour name="Austria" direction="E"/>
        <neighbour name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">...</country>
    <country name="Australia">...</country>
</countries>
```

# XML and JSON Data (Cont'd)

Example XML: `xml.etree.ElementTree`

```
import xml.etree.ElementTree as ET

tree = ET.parse('country_data.xml')
root = tree.getroot()
# displays 'countries'
root.tag
for c in root:
    print c.tag c.attrib
# displays ...
# country {'name': 'Liechtenstein'}
# country {'name': 'Singapore'}
# country {'name': 'Australia'}
```

# XML and JSON Data (Cont'd)

KML and Geographic Data: KeyHole Markup Languale (KML) [1] is a file format used to record geographic data. Currently it is under the purview of the Open Geospatial Consortium (OGC) [2].

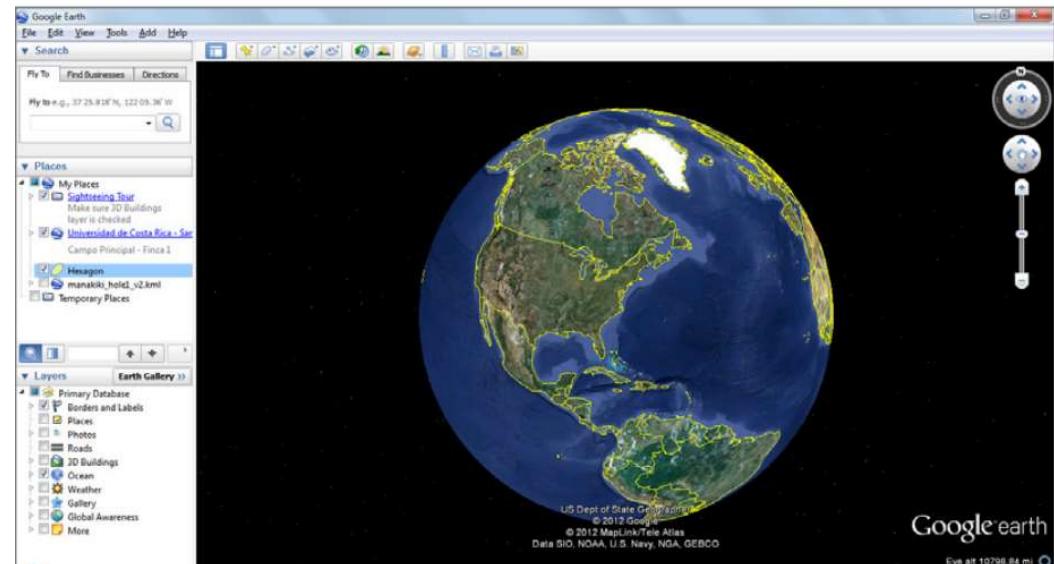
The goal of KML is to become the single international standard language for expressing geographic annotation and visualization on existing or future web-based online and mobile maps and earth browsers.

Example of applications using the format are:

Google Earth,

Google Maps, and

Google Maps for Mobile Apps



27 References: [1] Displaying KML files in your Maps application. <https://developers.google.com/maps/tutorials/kml/>  
[2] Open Geospatial Consortium. <http://www.opengeospatial.org/standards/kml#overview>

# XML and JSON Data (Cont'd)

```
<Placemark>
```

```
    <name>Cleveland State University</name>
```

```
    <ExtendedData>
```

```
        <Data name="video">
```

```
            <value><![CDATA[<iframe width="640" height="360"
src="http://www.youtube.com/embed/es9KEhVl0iw"
frameborder="0"
allowfullscreen></iframe><br><br>]]></value>
```

```
        </Data>
```

```
    </ExtendedData>
```

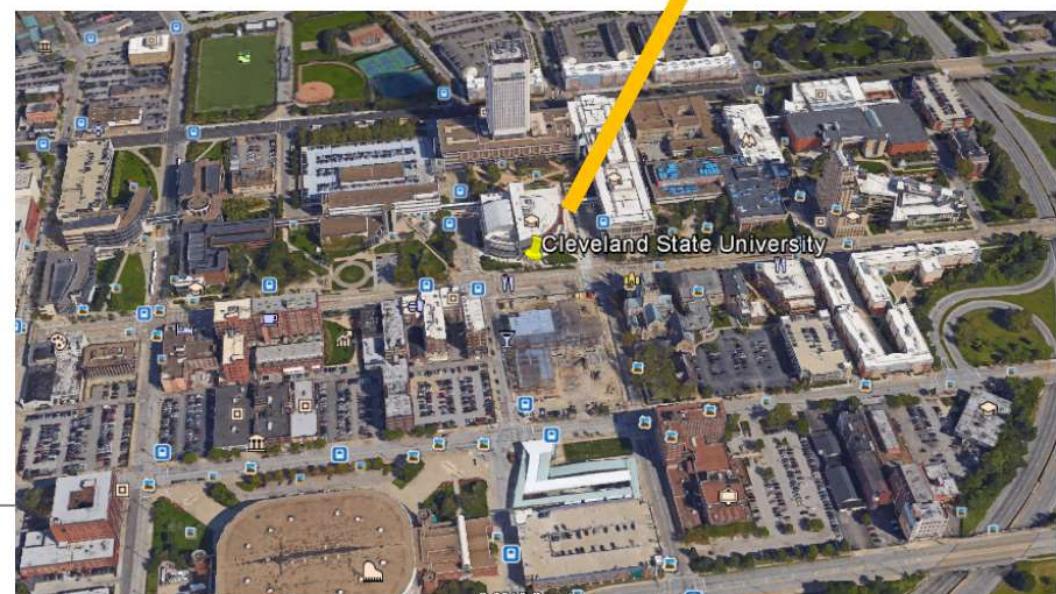
```
    <Point>
```

```
        <coordinates> -81.675281,
41.501792,
0
```

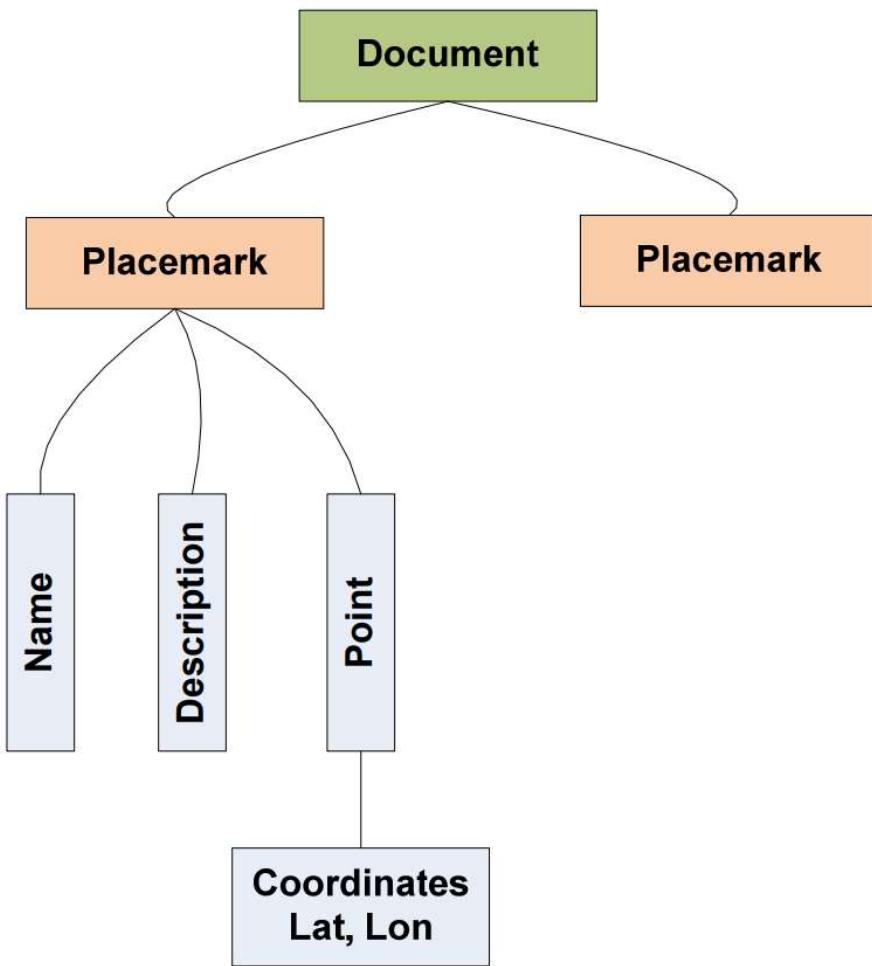
```
    </coordinates>
```

```
    </Point>
```

```
</Placemark>
```



# XML and JSON Data (Cont'd)



In this example a Document consists of various Placemark elements. The markers identify a set of points-of-interest.

Each of our includes a

- Name,
- Description,
- and
- Geo-Point including: latitude, longitude and altitude.

[http://code.google.com/apis/kml/documentation/kml\\_tut.html](http://code.google.com/apis/kml/documentation/kml_tut.html)

# XML and JSON Data (Cont'd)

JSON (JavaScript Object Notation) is a plain-text formatting protocol for encoding and decoding hierarchically structured data.

1. JSON is based on JavaScript Programming Language
2. It is language and platform independent.
3. Arguably, it is easier to read and write than XML.
4. A JSON encoded data-set is based on the manipulation of two common programming constructs: simple arrays and objects.
5. Each object is represented as an associative-array holding a collection of attributes and their values.
6. An attribute's value could be a simple data-type or another nested JSON object.

# XML and JSON Data (Cont'd)

| Object[ 0 ]  | Object[ 1 ]  |  | Object[ M ]  |
|--|--|--|--|
| {<br>Attr_1 : value1<br>...<br>Attr_n0 : value_n0<br>} | {<br>Attr_1 : value1<br>...<br>Attr_n1 : value_n1<br>} |  | {<br>Attr_1 : value1<br>...<br>Attr_nM : value_nM<br>} |

Individual data items are represented as key : value pairs

- Data items are separated by commas
- Objects are enclosed inside curly braces { }
- Arrays of objects are delimited by square brackets [ ]

```
"Person" :  
[  
    {"name": "Daenerys", "age": 20},  
    {"name": "Arya", "age": 12},  
    {"name": "Cersei", "age": 35}  
]
```

# XML and JSON Data (Cont'd)

Much useful data is available in JSON format

- » typically from web service API's

Python provides JSON library module (`json`)

- » `dump()` serializes Python objects as JSON data
  - » `load()` converts JSON data into Python objects
  - » apply standard Python methods to load'ed objects

Example: <https://hackerone.com/reports/328486.json>

# CSV Data

Most CSV data is effectively a (relational) table

- » however may not be normalised (in RDB sense)

Much CSV data is produced from spreadsheets

- » column headings provide metadata (if available)

Example:

```
1 Subject,Start Date,Start Time,End Date,End Time
2 TERM 1,2/18/2019,12:00:00 AM,5/2/2019,12:00:00 AM
3 Term 1 begins,2/18/2019,12:00:00 AM,2/19/2019,12:00:00 AM
4 "Term 1, Week 1",2/18/2019,12:00:00 AM,2/19/2019,12:00:00 AM
5 "Term 1, Week 2",2/25/2019,12:00:00 AM,2/26/2019,12:00:00 AM
6 "Term 1, Week 3",3/4/2019,12:00:00 AM,3/5/2019,12:00:00 AM
7 "Term 1, Week 4",3/11/2019,12:00:00 AM,3/12/2019,12:00:00 AM
8 Deadline for exam paper submission,3/14/2019,12:00:00 AM,3/15/2019,12:00:00 AM
9 CENSUS DATE,3/17/2019,12:00:00 AM,3/18/2019,12:00:00 AM
10 "Term 1, Week 5",3/18/2019,12:00:00 AM,3/19/2019,12:00:00 AM
11 "Term 1, Week 6",3/25/2019,12:00:00 AM,3/26/2019,12:00:00 AM
12 "Term 1, Week 7",4/1/2019,12:00:00 AM,4/2/2019,12:00:00 AM
13 Class registration opens for Term 2 classes,4/8/2019,12:00:00 AM,4/9/2019,12:00:00 AM
```

# CSV Data (Cont'd)

Python has several CSV modules

csv provides basic reading/writing of CSV data

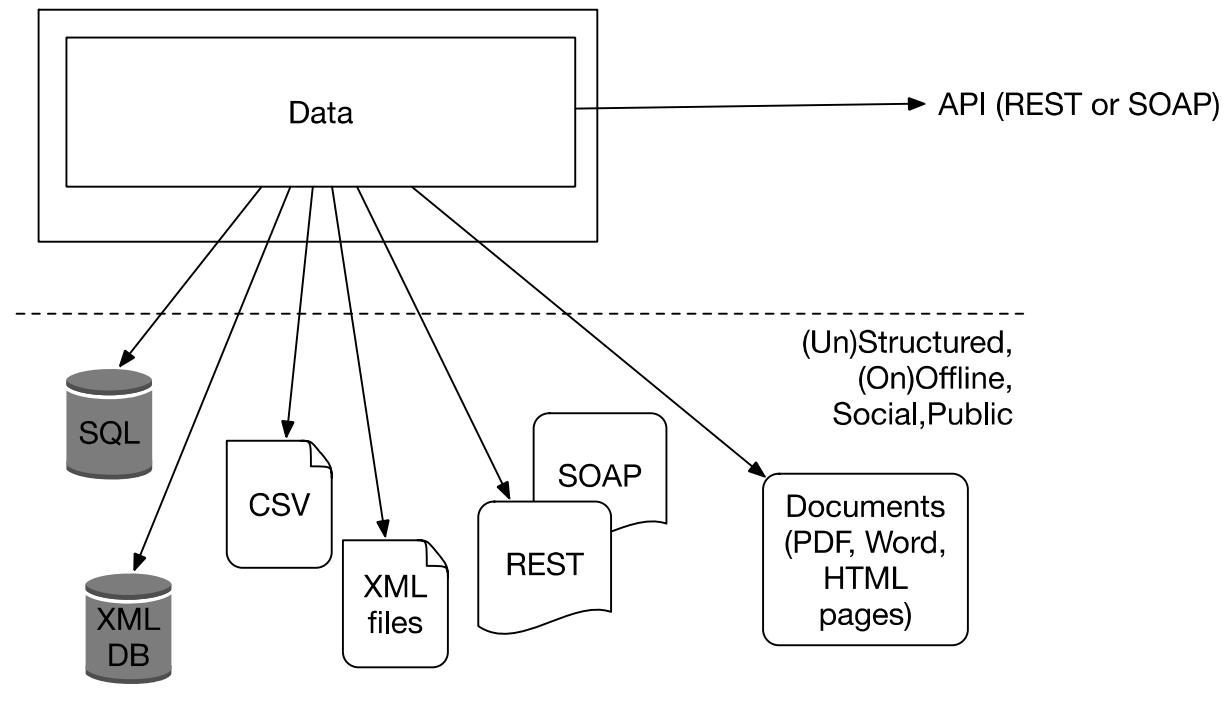
- » each row becomes a Python list
- » collection of rows is a list of lists

pandas provides reading/writing CSV data

- » also, an abstraction of the data (DataFrame)
- » plus a range of operators for filtering/calculating

Example: <http://www.abs.gov.au/browse?opendocument&ref=topBar>

# Back to Data Services



# Challenge

How do we store and access this data ?

E-Commerce website

- » Data operations are mainly transactions (Reads and “Writes”)
- » Operations are mostly on-line
- » Response time should be quick but important to maintain security and reliability of the transactions.
- » ACID properties are important

Why?

A screenshot of a web-based hotel search form. At the top is a search bar labeled "Destination" with placeholder text "e.g. city, region, district or specific hotel". Below it are fields for "Check-in" and "Check-out", each consisting of a date selector (calendar icon) followed by dropdown menus for "Day" and "Month". There is also a checkbox "I don't have specific dates yet". Underneath these are fields for "Guests" (set to "2 adults in 1 room") and a large green "Search" button. At the bottom of the form, there is a link "More search options" and a list of popular destinations: Melbourne, Canberra, Sydney, Gold Coast, Honolulu, Surfers Paradise, Wollongong, Patong, Singapore, Waikiki, Las Vegas, and Brisbane.

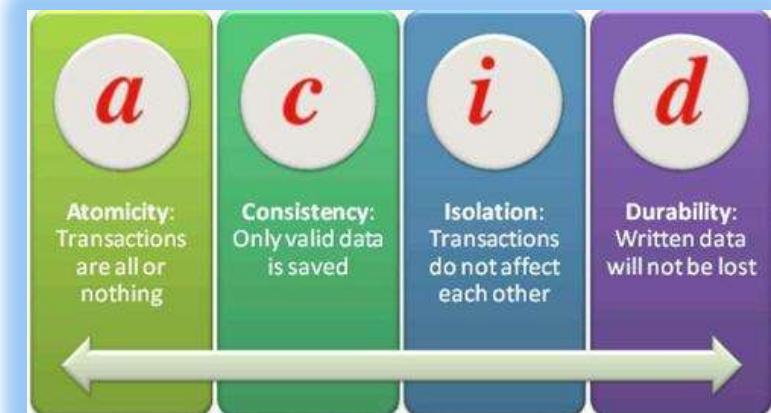


# Challenge

Image serving website (many social network sites in general)

Data operations are mainly fetching information (Reads)  
– also the “*fan-out*” effect is challenge

Operations are mainly on-line  
High bandwidth requirement  
ACID requirements can be relaxed



# Challenge

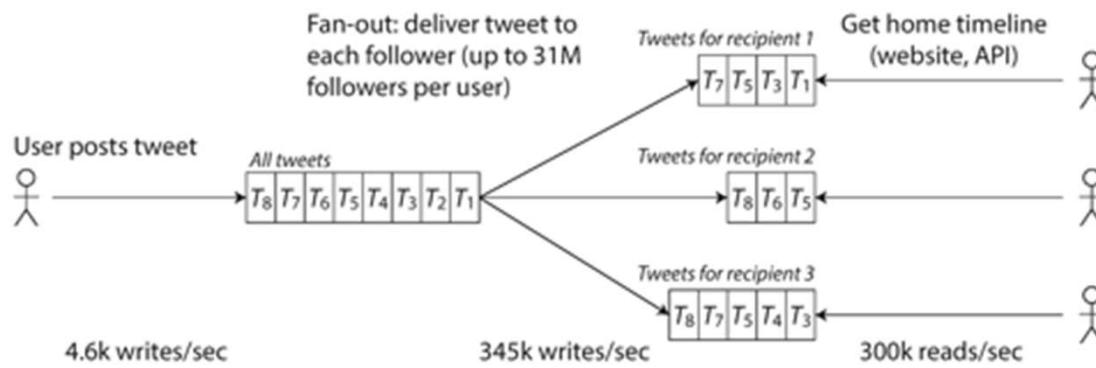


Figure 1-3. Twitter's data pipeline for delivering tweets to followers, with load parameters as of November 2012 [16].

A user can see tweets posted by the people they follow ...

- A new post  $\rightarrow$  look up the followers and ‘write’ to each follower’s timeline ahead of time  $\rightarrow$  makes reading easy
- But this also creates a lot of ‘writing’ work
  - On average 75 followers, but can vary widely (some users have 30 million followers)
  - May need to consider the distribution of the followers per user (and how often each user tweets)

# Challenge

## Search Engines

- Data operations are mainly reading index files for answering queries (Reads)
- Index compilation is performed off-line due to the large size of source data (the entire Web)
- ACID requirements can be relaxed
- Response times must be as fast as possible.



# Challenge for API ...

How do we store and access this data over the web ?

- » Consumption of Data (for you to take data in ...)
- » Publication of Data (for you to make data available for others ...)

Important question: What is your data model behind the API?

Data models can change how we think about the problem you are solving

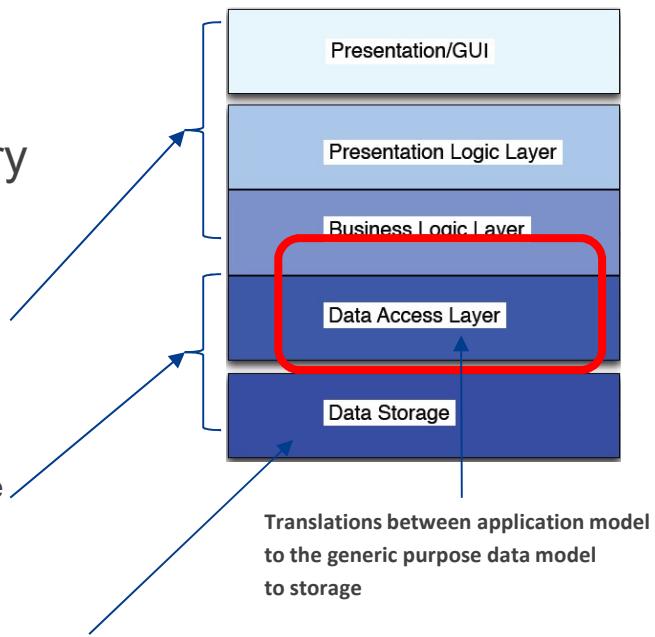
# What is in a data model ...

An application developer “thinks” in terms of the real world (people, organisations, actions, goods, etc.) ... and model it as objects/data structures and APIs that manipulate them – these models are very specific to each application

When you want to store the objects, you express them in generic-purpose data model such as JSON, XML documents or tables.

The “storage” also allows the representation to be queried, searched or manipulated.

The engineers of the ‘storage solution’ software decide on how JSON/XML/tables are represented in terms of bytes in memory, disk or on a network.



# Relational Model vs. “NoSQL” Models

Relational Model (more or so synonymous with SQL)

The best known, probably the most successful data model which has proven itself in many aspects to be the data model of choice in many applications

Data is organised into relations (table) where each relation holds an unordered collection of tuples (rows)

Based on solid theory and well engineered implementation -> many competing models have been proposed, but never managed to take over SQL

Built for business data processing

- Typical business transactions (airline reservations, stock keeping, etc.)
- Batch processing (invoicing, payroll, reporting, etc.)

Turned out it was still generically applicable to many modern Web applications too

Hypothetical Relational Database Model

| PubID      | Publisher         | PubAddress               |
|------------|-------------------|--------------------------|
| 03-4472822 | Random House      | 123 4th Street, New York |
| 04-7733903 | Wiley and Sons    | 45 Lincoln Blvd, Chicago |
| 03-4859223 | O'Reilly Press    | 77 Boston Ave, Cambridge |
| 03-3920886 | City Lights Books | 99 Market, San Francisco |

| AuthorID    | AuthorName     | AuthorBDay |
|-------------|----------------|------------|
| 345-28-2938 | Haile Selassie | 14-Aug-92  |
| 392-48-9965 | Joe Blow       | 14-Mar-15  |
| 454-22-4012 | Sally Hemmings | 12-Sep-70  |
| 663-59-1254 | Hannah Arendt  | 12-Mar-06  |

| ISBN          | AuthorID    | PubID      | Date | Title                       |
|---------------|-------------|------------|------|-----------------------------|
| 1-34532-482-1 | 345-28-2938 | 03-4472822 | 1990 | Cold Fusion for Dummies     |
| 1-38482-995-1 | 392-48-9965 | 04-7733903 | 1985 | Macrame and Straw Tying     |
| 2-35921-499-4 | 454-22-4012 | 03-4859223 | 1852 | Fluid Dynamics of Aquaducts |
| 1-38279-293-4 | 663-59-1254 | 03-3920886 | 1967 | Beads, Baskets & Revolution |

Image: [http://gnosis.cx/publish/programming/xml\\_matters\\_8.html](http://gnosis.cx/publish/programming/xml_matters_8.html)

# Relational Model vs. “NoSQL” Models

## The rise of NoSQL ... (since 2010 or so)

Refers to a host of technologies that implement distributed, “non-relational” databases

## Why NoSQL?

- A need for greater scalability – very large datasets or very high ‘write’ throughput
  - A need for more expressive and dynamic data model
  - Usually do not require a fixed table schema nor do they use the concept of joins
  - All NoSQL offerings relax one or more of the ACID properties

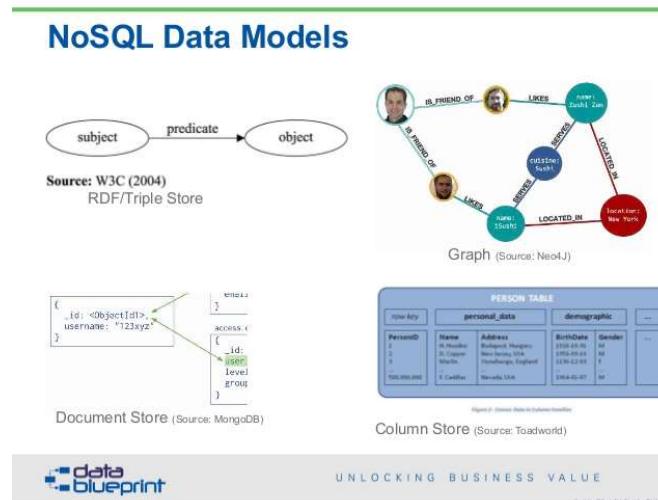


Image: <https://www.slideshare.net/Dataversity/trends-in-data-modeling>

# Problems with Relational Models

Normalisation ... 3NF

\* many fragments -> leading to many joins -> scalability ?

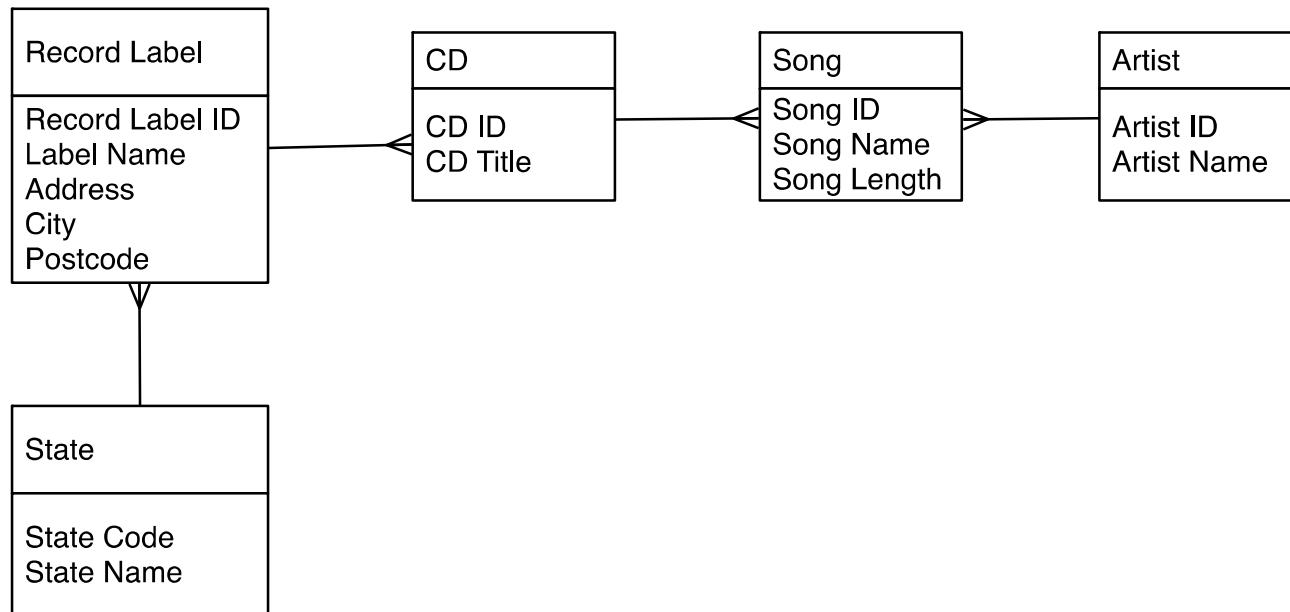


Image: <https://www.slideshare.net/Dataversity/trends-in-data-modeling>

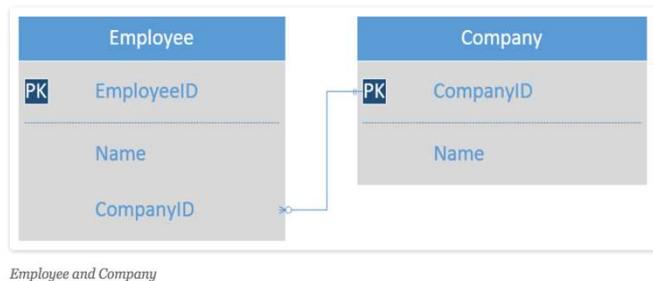
# Problems with Relational Models

The Object-Relational Mismatch (Impedance Mismatch)

Refers to the problem of a mismatch between application data model (your business objects) and data model for storage (in relational tables)

The object-relational impedance mismatch arises when to combine an object-oriented code with a relational database.

This mismatch creates a need for an awkward translation layer between the objects in the application code and the database model of tables/row/columns.



```
public class Employee
{
    public string Name { get; private set; }
    public Company Company { get; private set; }
}

public class Company
{
    public string Name { get; private set; }
    public List<Employee> Employees { get; private set; }
}
```

It's now your responsibility as a developer to maintain both ends: the reference on one side and the collection of references on the other.

# Alternative Data Models?

Relational Modelling  
of a resume  
(e.g., LinkedIn Profile)

Typical normalised form  
would put multi-values in  
separate tables with  
user\_id  
as foreign key

Fragmented tables -> join

(Chapter 2)

<http://www.linkedin.com/in/williamhgates>

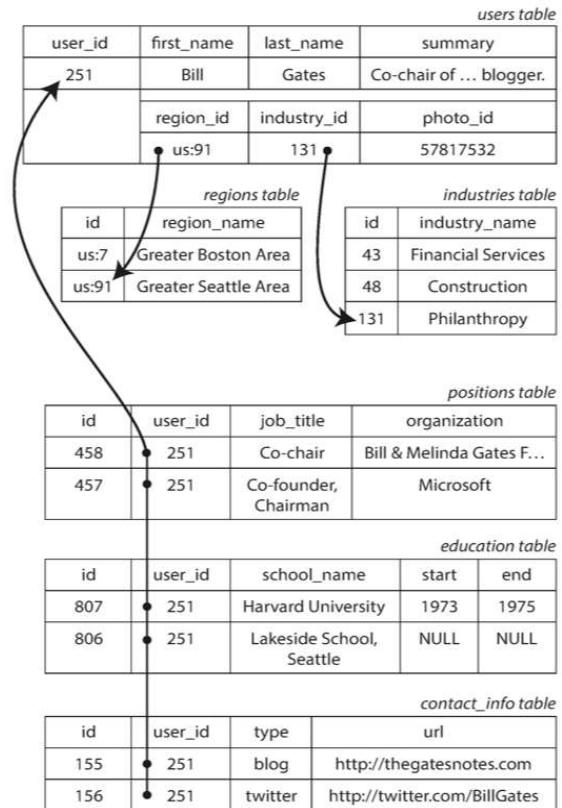
 **Bill Gates**  
Greater Seattle Area | Philanthropy

**Summary**  
Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

**Experience**  
Co-chair • Bill & Melinda Gates Foundation  
*2000 – Present*  
Co-founder, Chairman • Microsoft  
*1975 – Present*

**Education**  
Harvard University  
*1973 – 1975*  
Lakeside School, Seattle

**Contact Info**  
Blog: [thegatesnotes.com](http://thegatesnotes.com)  
Twitter: @BillGates

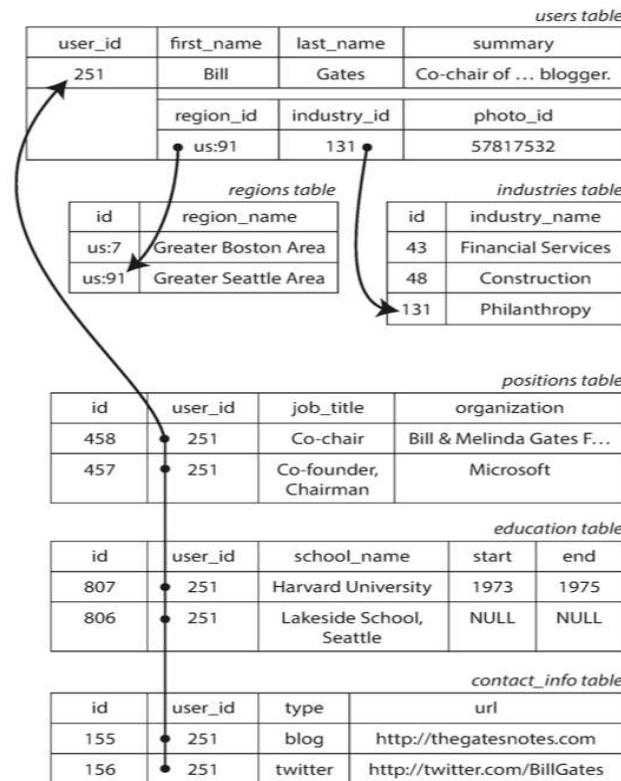


# Added features in SQL ...

Some databases support an idea similar to 'Arrays':

- can store multi values in a single row
- can be queried and indexed

| User_id | ... | Job_title  | School_name  |
|---------|-----|--|--|
| 251     |     | {co-chair, Bill & Melinda Gates ..}, {Chairman, Microsoft} | {Harvard University, 1973,1975},{Lake side School, Null, Null} |



(Chapter 2)

# Alternative Data Models?

Another option:

Encodes jobs, education, contact info  
as a JSON (or XML) document

Stores the whole document in a text  
column in the database

Application code accessing this info  
will have to deal with the structure as  
a whole

You cannot use the database to query  
for values inside the column

*Example 2-1. Representing a LinkedIn profile as a JSON document*

```
{  
    "user_id": 251,  
    "first_name": "Bill",  
    "last_name": "Gates",  
    "summary": "Co-chair of the Bill & Melinda Gates...  
Active blogger.",  
    "region_id": "us:91",  
    "industry_id": 131,  
    "photo_url": "/p/7/000/253/05b/308dd6e.jpg",  
    "positions": [  
        {"job_title": "Co-chair", "organization": "Bill &  
        Melinda Gates Foundation"},  
        {"job_title": "Co-founder, Chairman", "organization":  
        "Microsoft"}  
    ],  
    "education": [  
        {"school_name": "Harvard University", "start":  
        1973, "end": 1975},  
        {"school_name": "Lakeside School, Seattle", "start":  
        null, "end": null}  
    ],  
    "contact_info": {  
        "blog": "http://thegatesnotes.com",  
        "twitter": "http://twitter.com/BillGates"  
    }  
}
```

Document-based databases support this idea naturally  
(e.g., MongoDB – insert/query JSON objects)

# Document-based databases

MongoDB (the most well-known example)

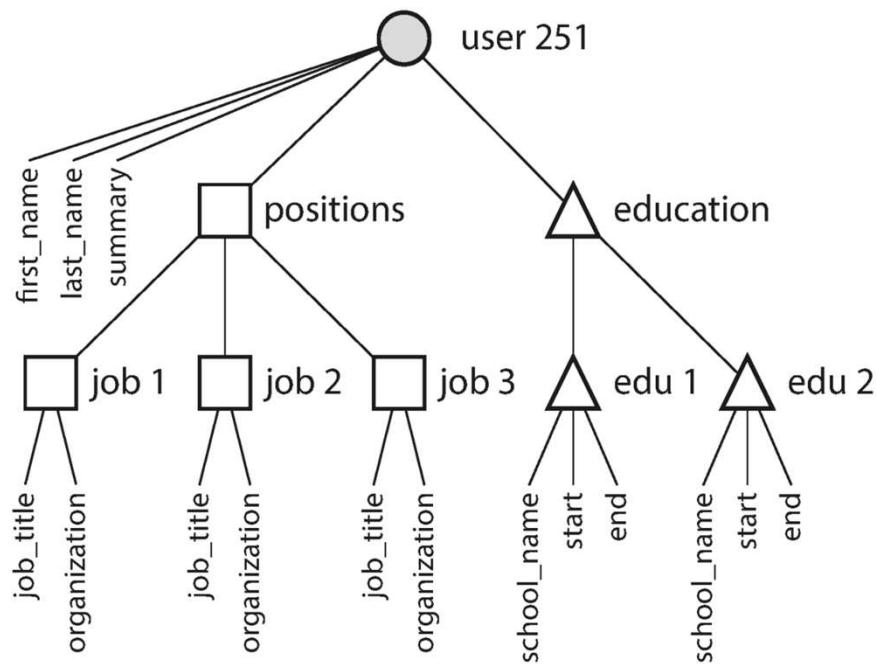
| RDBMS       | MongoDB   |
|-------------|---|
| Database    | Database  |
| Table       | Collection  |
| Tuple/Row   | Document  |
| column      | Field   |
| Table Join  | Embedded Documents  |
| Primary Key | Primary Key (Default key <code>_id</code> provided by mongodb itself) |

## Notable points:

- Collections do not enforce a schema. Documents within a collection can have different fields. Typically, all documents in a collection are of similar or related purpose
- No joins (everything embedded in a single object)

[https://www.tutorialspoint.com/mongodb/mongodb\\_overview.htm](https://www.tutorialspoint.com/mongodb/mongodb_overview.htm)

# Document-based databases



Embedded objects normally are the result of One-to-Many relationships

Improved “locality”

- a single retrieval request is enough to get all necessary into one “User”

The mismatch between application data model and storage-purpose data model is significantly reduced

- “Create a User” (JSON) in app code and “Insert a User” (JSON) into Document Collections

# Document model is not good with ...

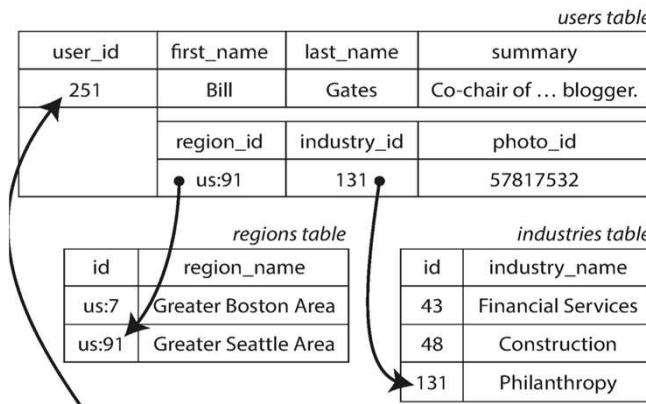
<http://www.linkedin.com/in/williamhgates>



**Bill Gates**  
Greater Seattle Area | Philanthropy

**Summary**  
Co-chair of the Bill & Melinda Gates Foundation.  
*Chairman Microsoft Corporation Vicechair*

What about Many-to-One or Many-to-Many?



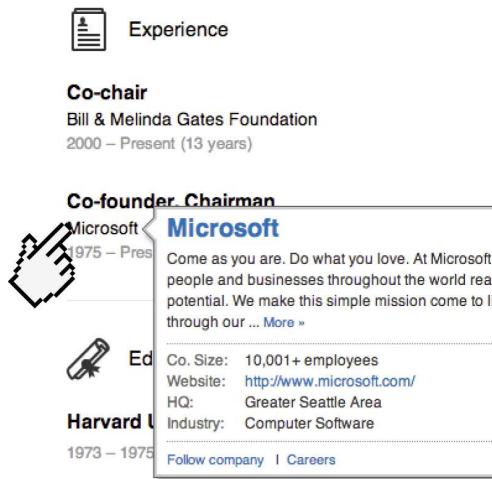
The relational model based solution of these “look-up tables” are useful:

- Consistent style and spelling across Users
- Avoiding ambiguity (e.g., if several cities with the same name)
- Ease of updating (the name is stored in only one place)
- Better search – a search for philanthropists in the state of Washington can match this User 251 (via another table)

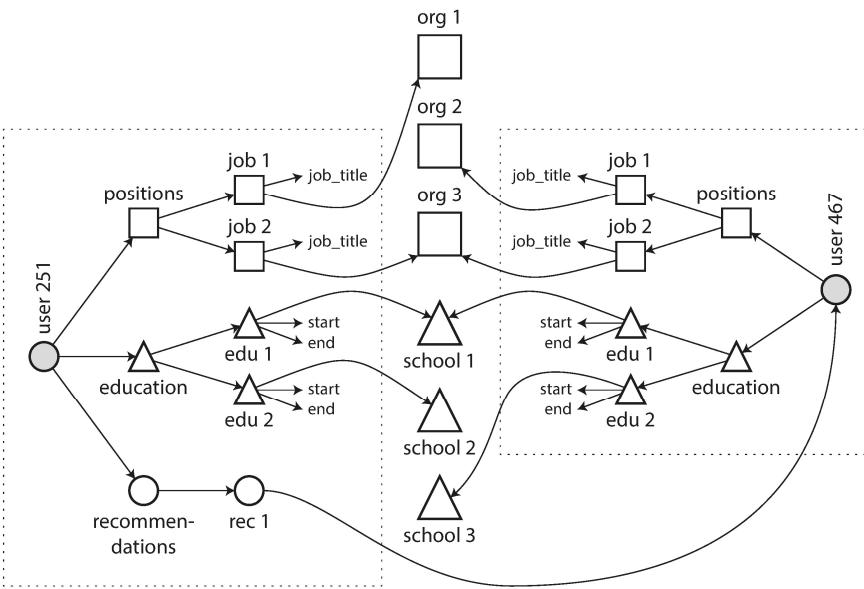
Storing ID vs Text -> NOT duplicating text is more flexible and keeps data consistent – reason for normalising in RDB

# Document model is not good with ...

The single “documents” tend to become more interconnected as more features are added



The company – linking it as a full entity by itself  
**(Many-to-One Relationship)**



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

# Relational vs. Document

When it comes to representing many-to-one and many-to-many relationships, both are not that different ...

- » Foreign keys (ID references) in relational
- » Document references (Doc ID) in document-based
  
- » The IDs are resolved at retrieval time by using a join or follow-up queries.
- » But joins on M-M or M-1 relationship are a routine – highly optimised at the database level
- » Document models – join support could be weak, application code might have to resolve the relationships as needed

# Relational vs. Document

Which data model leads to simpler application code?

- » If the application data model looks like a tree (document-like) -> it can be loaded at once using document-based model
- » If M-M relationships are central to the application data model -> relational model is efficient in joins. If document model is used, some of the 'join' logic will have to move to application code

Consider the kinds of relationships between data items. If they are highly interconnected data (e.g., social network)

- » document model is not so good,
- » relational model is OK ...
- » graph models would be natural (to be seen later)

# Relational vs. Document

Schema flexibility, always a good thing?

Most document-based databases do not enforce any schema in documents  
(schema-less databases)

- » Arbitrary keys and values can be added to a document and when reading clients have no guarantees as to what fields the documents may contain

Schema-on-read

- » The structure of the data is implicit, only interpreted when the data is read by application code
- » ≈ dynamic (runtime) type checking
- Schema-on-write
  - » The traditional approach of RDB - explicit schema and the database ensures all written data conforms to it
  - » ≈ static (compile-time) type checking

# Relational vs. Document

# Schema flexibility, always a good thing?

When does this ‘schema-on-read/write’ matter? -> when application wants to change the format of its data.

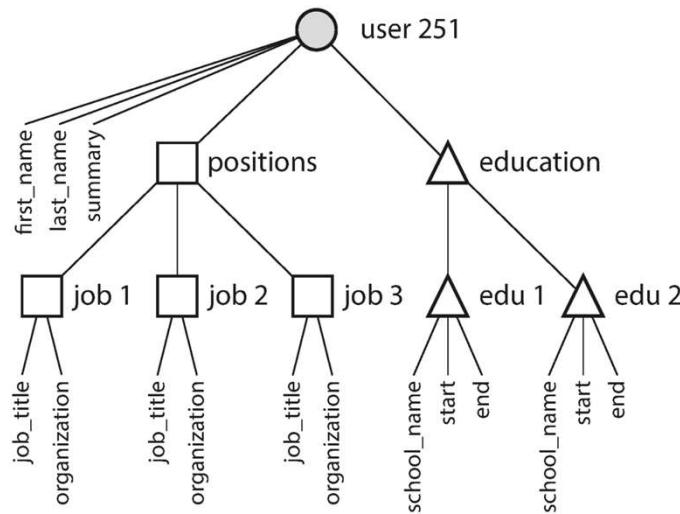
E.g., User name in one field -> User name in two fields.

```
if (user && user.name && !user.first_name) {  
    // Documents written before Dec 8, 2013 don't have  
    first_name  
    user.first_name = user.name.split(" ")[0];  
}  
  
ALTER TABLE users ADD COLUMN first_name text;  
UPDATE users SET first_name = split_part(name, ' ', 1);  
-- PostgreSQL  
UPDATE users SET first_name = substring_index(name, ' ',  
1); -- MySQL
```

DOC model is considered advantageous if the docs in the collection tend to have different structures (e.g., different types of related objects)

# Relational vs. Document

Data locality for queries - doc-based systems store a document as a single continuous string as JSON or XML (or a binary variant)



If your application requires the entire document (e.g., to render it on a Web page as a whole), there is a performance advantage over split tables

The diagram illustrates a relational database schema with five tables:

- users table:**

| user_id | first_name | last_name   | summary                  |
|---------|------------|-------------|--------------------------|
| 251     | Bill       | Gates       | Co-chair of ... blogger. |
|         | region_id  | industry_id | photo_id                 |
|         | us:91      | 131         | 57817532                 |
- regions table:**

| id    | region_name          |
|-------|----------------------|
| us:7  | Greater Boston Area  |
| us:91 | Greater Seattle Area |
- industries table:**

| id  | industry_name      |
|-----|--------------------|
| 43  | Financial Services |
| 48  | Construction       |
| 131 | Philanthropy       |
- positions table:**

| id  | user_id | job_title            | organization              |
|-----|---------|----------------------|---------------------------|
| 458 | 251     | Co-chair             | Bill & Melinda Gates F... |
| 457 | 251     | Co-founder, Chairman | Microsoft                 |
- education table:**

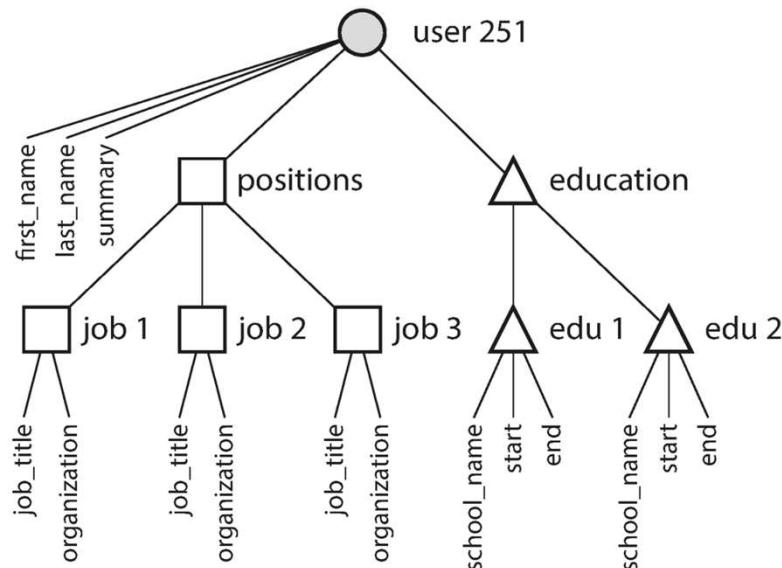
| id  | user_id | school_name              | start | end  |
|-----|---------|--------------------------|-------|------|
| 807 | 251     | Harvard University       | 1973  | 1975 |
| 806 | 251     | Lakeside School, Seattle | NULL  | NULL |
- contact\_info table:**

| id  | user_id | type    | url   |
|-----|---------|---------|---|
| 155 | 251     | blog    | <a href="http://thegatesnotes.com">http://thegatesnotes.com</a>         |
| 156 | 251     | twitter | <a href="http://twitter.com/BillGates">http://twitter.com/BillGates</a> |

Arrows indicate relationships between the user node and the other tables. Specifically, the user node connects to the user\_id column in the positions, education, and contact\_info tables. The regions table connects to the region\_id column in the users table. The industries table connects to the industry\_id column in the users table. The positions table connects to the user\_id column in the education table. The contact\_info table connects to the user\_id column in the education table.

# Relational vs. Document

The locality of Doc-based systems

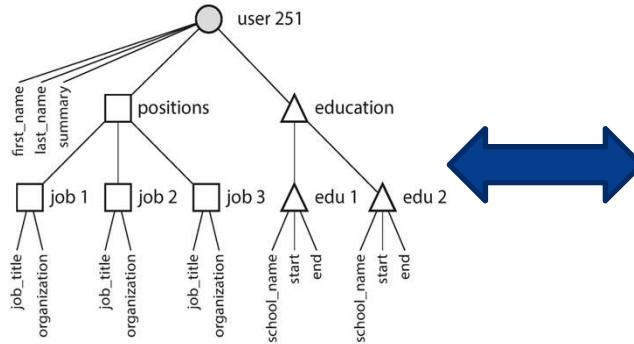


Data locality advantage only applies only if you need large parts of the document at a time (often the whole document needs to be loaded only if you need to access a small portion of it)

On Updates, normally the whole document needs to be rewritten (except tiny changes that do not change the overall encoded size of the document)

# Relational vs. Document

Convergence of document and relational databases



Hypothetical Relational Database Model

| PubID      | Publisher         | PubAddress               |
|------------|-------------------|--------------------------|
| 03-4472822 | Random House      | 123 4th Street, New York |
| 04-7733903 | Wiley and Sons    | 45 Lincoln Blvd, Chicago |
| 03-4859223 | O'Reilly Press    | 77 Boston Ave, Cambridge |
| 03-3920886 | City Lights Books | 99 Market, San Francisco |

| AuthorID    | AuthorName     | AuthorBDay |
|-------------|----------------|------------|
| 345-28-2938 | Haile Selassie | 14-Aug-92  |
| 392-48-9965 | Joe Blow       | 14-Mar-15  |
| 454-22-4012 | Sally Hemmings | 12-Sep-70  |
| 663-59-1254 | Hannah Arendt  | 12-Mar-06  |

| ISBN          | AuthorID    | PubID      | Date | Title                       |
|---------------|-------------|------------|------|-----------------------------|
| 1-34532-482-1 | 345-28-2938 | 03-4472822 | 1990 | Cold Fusion for Dummies     |
| 1-38482-995-1 | 392-48-9965 | 04-7733903 | 1985 | Macrame and Straw Tying     |
| 2-35921-499-4 | 454-22-4012 | 03-4859223 | 1852 | Fluid Dynamics of Aquaducts |
| 1-38278-293-4 | 663-59-1254 | 03-3920886 | 1967 | Beads, Baskets & Revolution |

PostgreSQL (since v.9.3), MySQL (since v.5.7). IBM DB2 (since v.10) support JSON documents.

RethinkDB, MongoDB (document-based) support relational-like joins in its query language

The two models can complement each other -> A hybrid model seems like a trend in these two systems

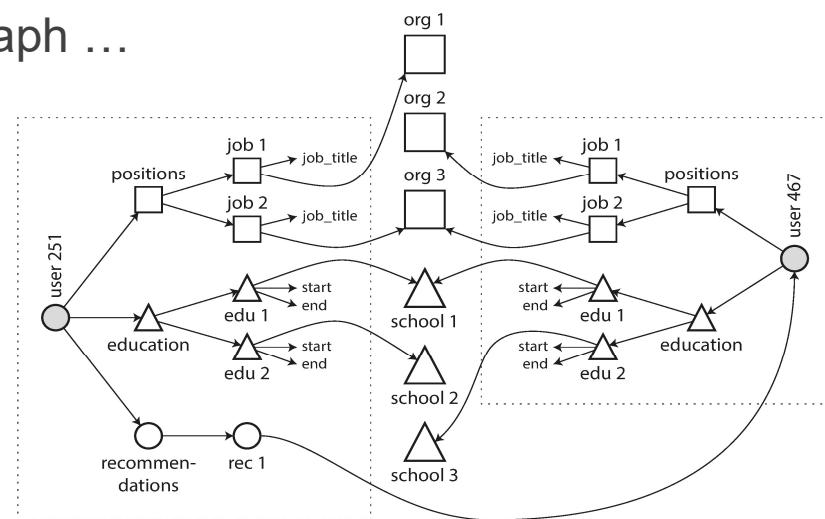
<https://www.zdnet.com/article/the-emergence-of-nosql-and-convergence-with-relational-databases/>

# 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 ...

Graph:

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



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



# Graph-like Models

Many kinds of data can be modelled as a graph

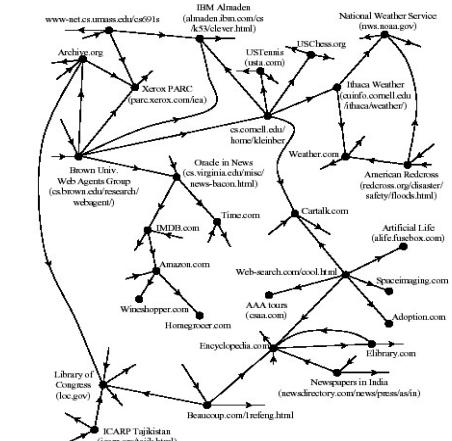
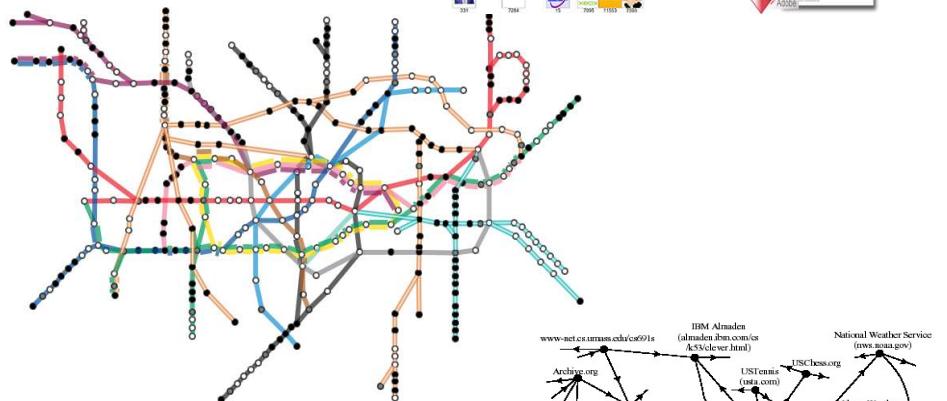
Social Graph – vertices are people, edges indicate which people know each other

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

61

Well-known algorithms on the model

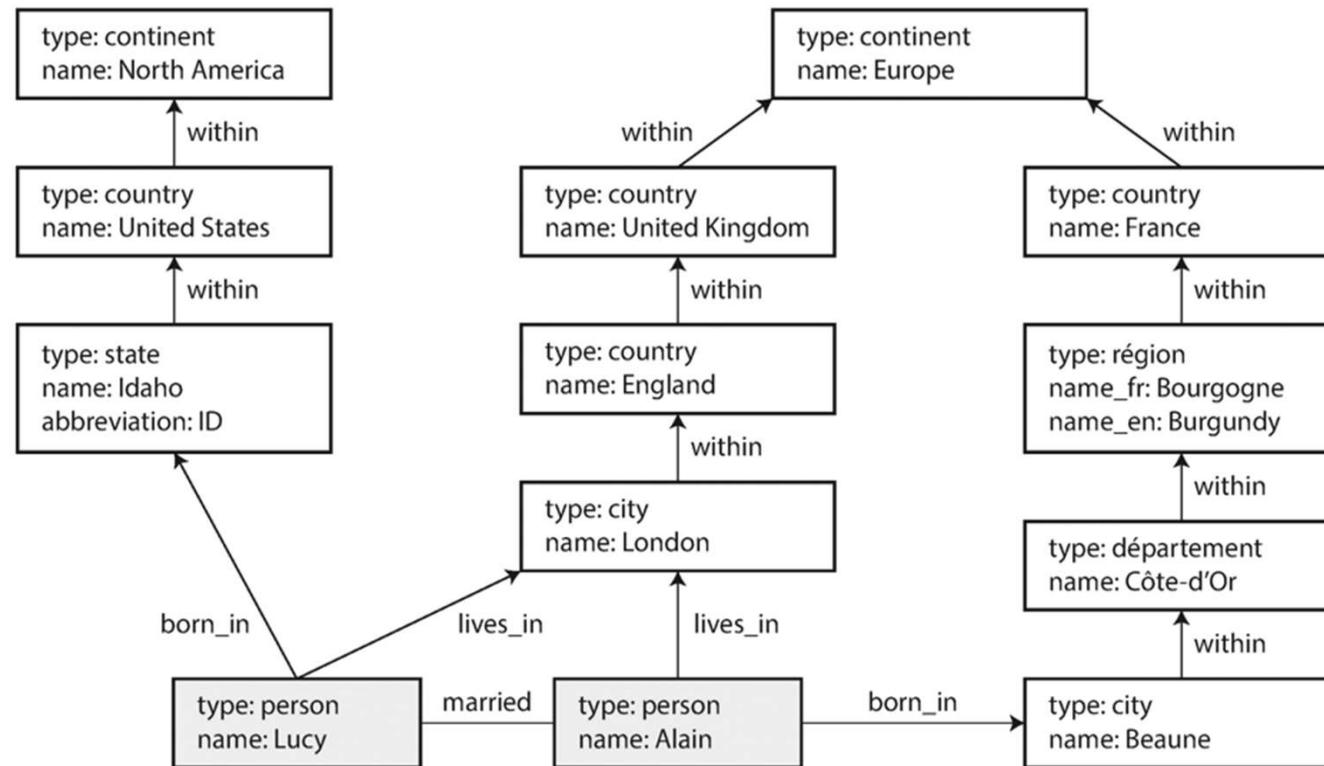


[http://www.supplychain247.com/article/why\\_supply\\_chains\\_should\\_be\\_more\\_socially\\_engaged](http://www.supplychain247.com/article/why_supply_chains_should_be_more_socially_engaged)  
<https://visualign.wordpress.com/2012/07/11/london-tube-map-and-graph-visualizations/>  
<http://canacopegl.com/single.php?id=http://www-inst.eecs.berkeley.edu/~cs61bl/r//cur/graphs/web.graph.png>

61

# Graph-like Models

Vertices are not limited to the same type of data.



# Graph-like Models

Facebook, TAO system (2013)

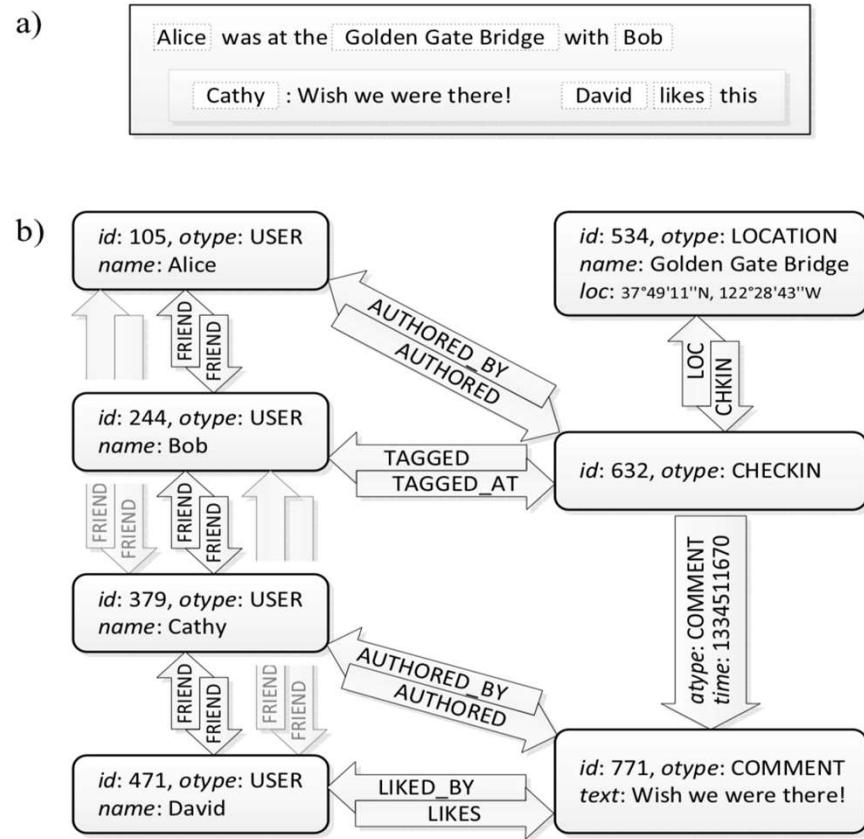


Figure 1: A running example of how a user's checkin might be mapped to objects and associations.

# 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 outgoing 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*

---

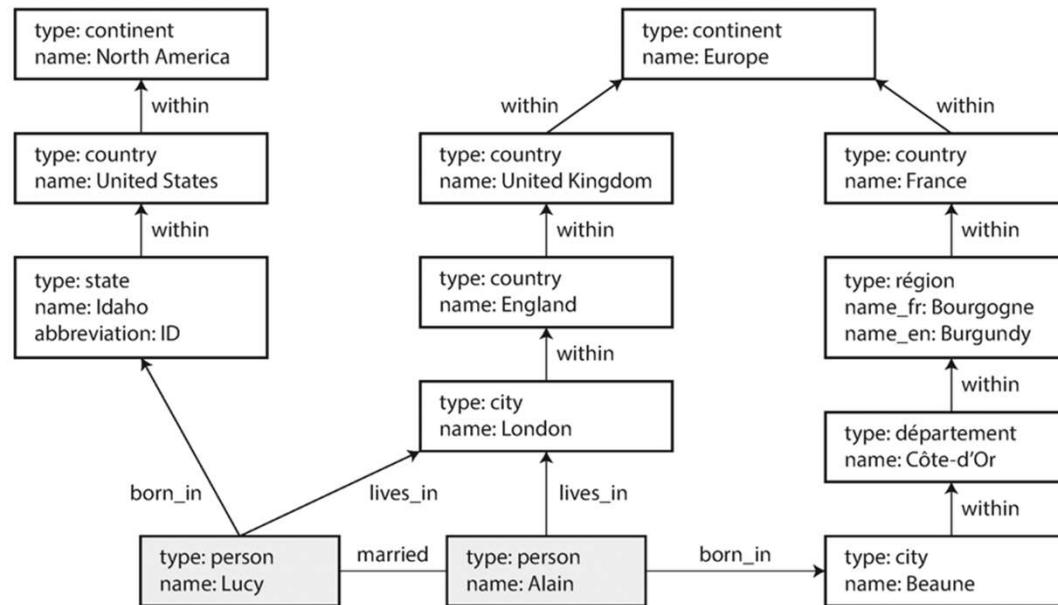
```
CREATE TABLE vertices (
    vertex_id    integer PRIMARY KEY,
    properties   json
);

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)

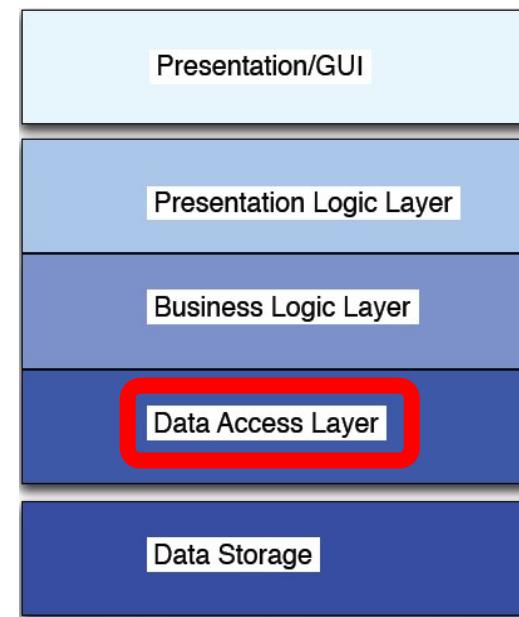
- Different kinds of regional structures in different countries
- Type country “within” a type country
- Varying granularity (e.g., born\_in “type:state”, lives\_in type:city)



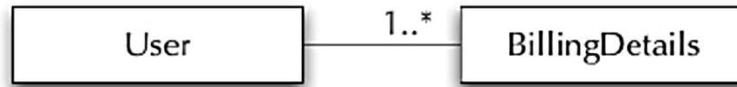
# Accessing DB from an application ...

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 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  
}
```

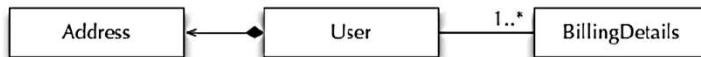
```
create table User (  
    username varchar(15)  
        not null primary key,  
    name varchar(50) not null,  
    address varchar(100)  
)
```

```
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:

Objects can be either equal or identical:

identical = same object (address)

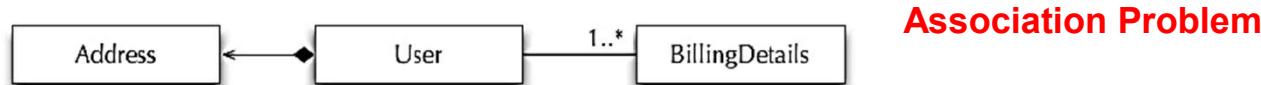
equal = same values

**Identity Concept Mismatch**

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

In OO, method chaining like:

## Object Graph Navigation

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
```

# Query languages for data

```
function getSharks() {  
    var sharks = [];  
    for (var i = 0; i < animals.length; i++) {  
        if (animals[i].family === "Sharks") {  
            sharks.push(animals[i]);  
        }  
    }  
    return sharks;  
}
```

```
SELECT * FROM animals WHERE family = 'Sharks';
```

Most programming languages are imperative:

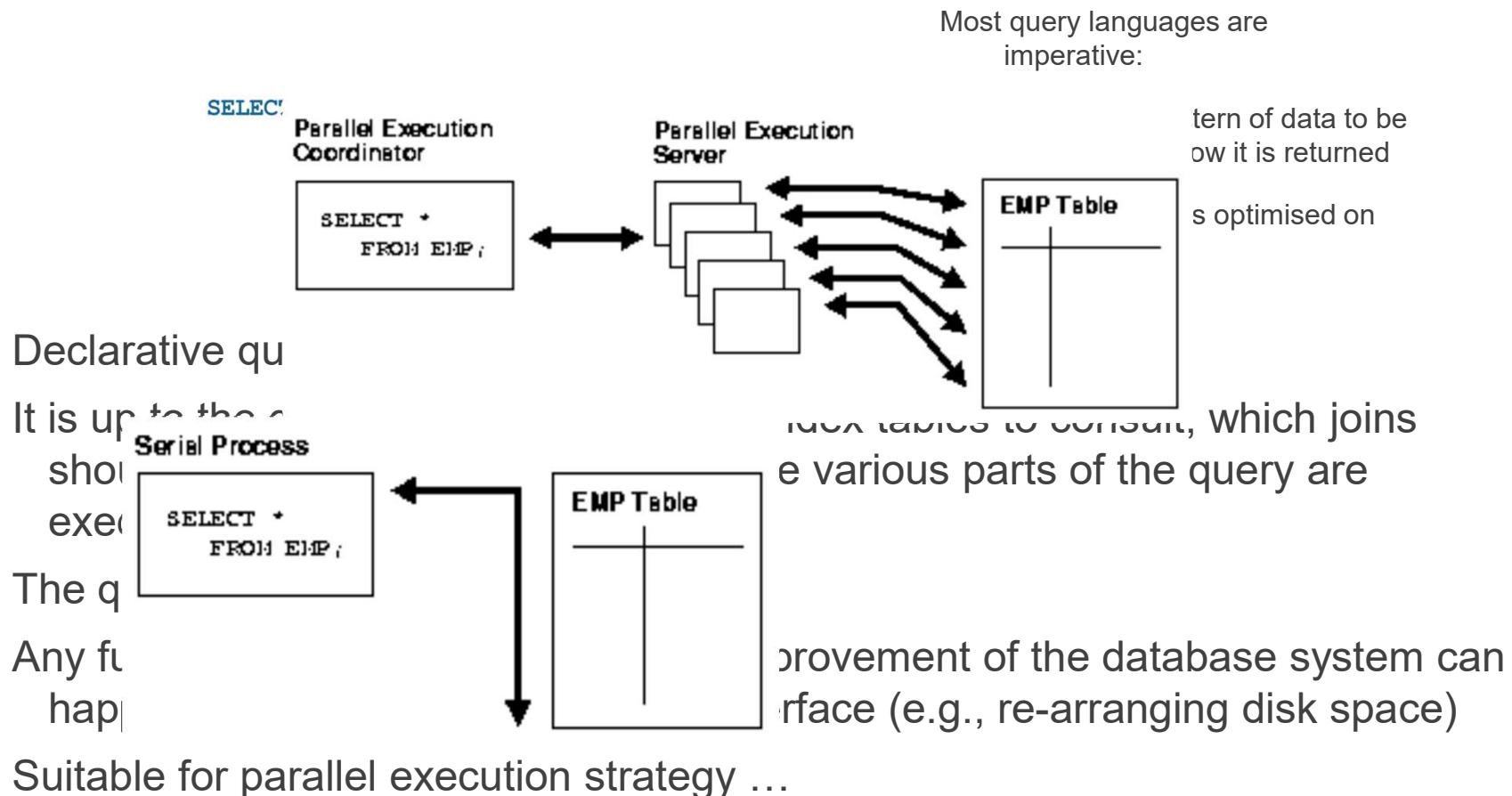
- step-by-step instructions on how the data should be returned ...

Most query languages are imperative:

- Specify the pattern of data to be returned, not how it is returned
- The database is optimised on how to do this

This declarative query paradigm is the same in Relational or Document-based systems

# Query languages for data



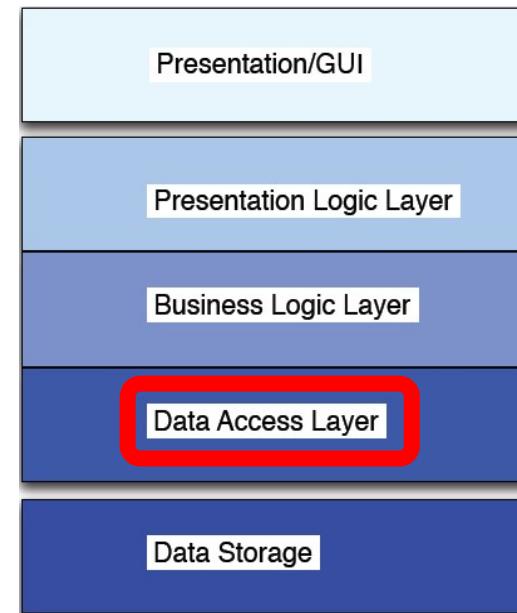
# Query Languages for Data

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”

Database connectivity API specifications

- Java has JDBC API, Python has DB-API, Microsoft variety has ODBC API, etc.
- Each specification is then implemented by the database system provider as a library for the developers (e.g., DB-API library for PostgreSQL, or JDBC library for Oracle)
- The application code blocks relating to using this library form “Data Access Layer” in the stack.



# Directly Executing SQL

DBAPI – e.g., psycopg2

```
import psycopg2
connection = psycopg2.connect("scott", "tiger", "test")

cursor = connection.cursor()
cursor.execute(
    "select emp_id, emp_name from employee "
    "where emp_id=%(emp_id)s",
    {'emp_id':5})
emp_name = cursor.fetchone()[1]
cursor.close()

cursor = connection.cursor()
cursor.execute(
    "insert into employee_of_month "
    "(emp_name) values (%(emp_name)s)",
    {"emp_name":emp_name})
cursor.close()

connection.commit()
```

Connect  
(network/or file handle)

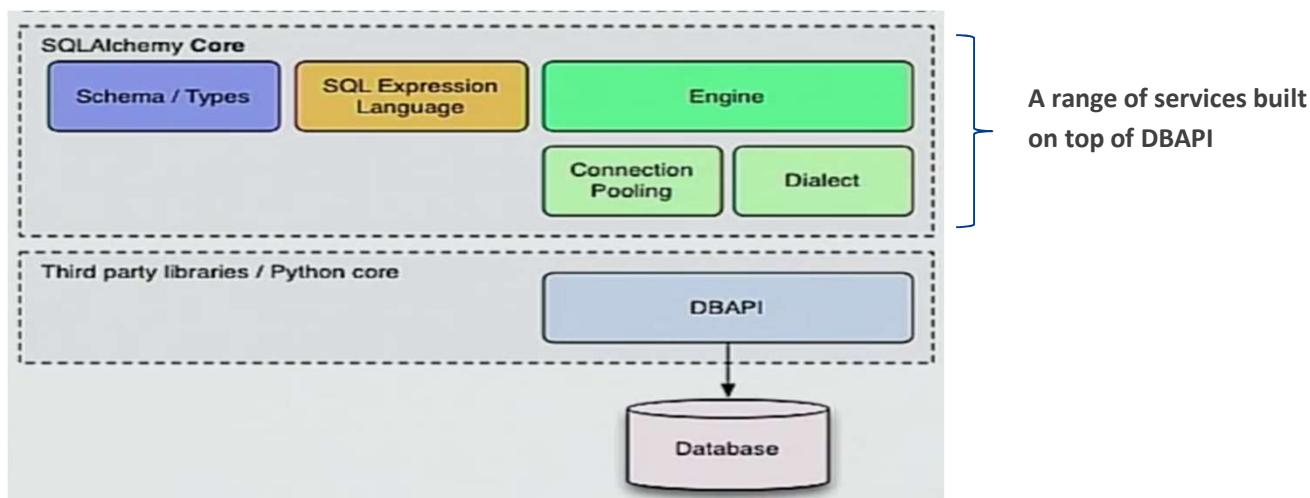
"An object for  
Table rows and loops  
within them"

Bound parameter

# Directly Executing SQL

## DBAPI

- » Many different implementations of the spec ...
- » Inconsistency between different implementations (e.g., bound parameter formats, exception hierarchy)
- » No explicit transaction markers (no begin() transaction)

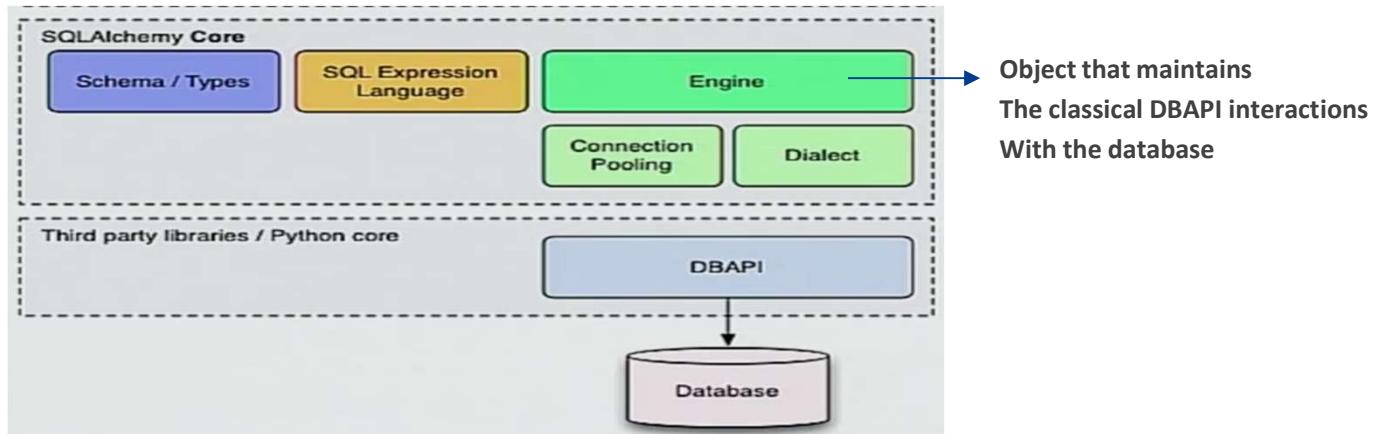


# Directly Executing SQL

## SQLAlchemy

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

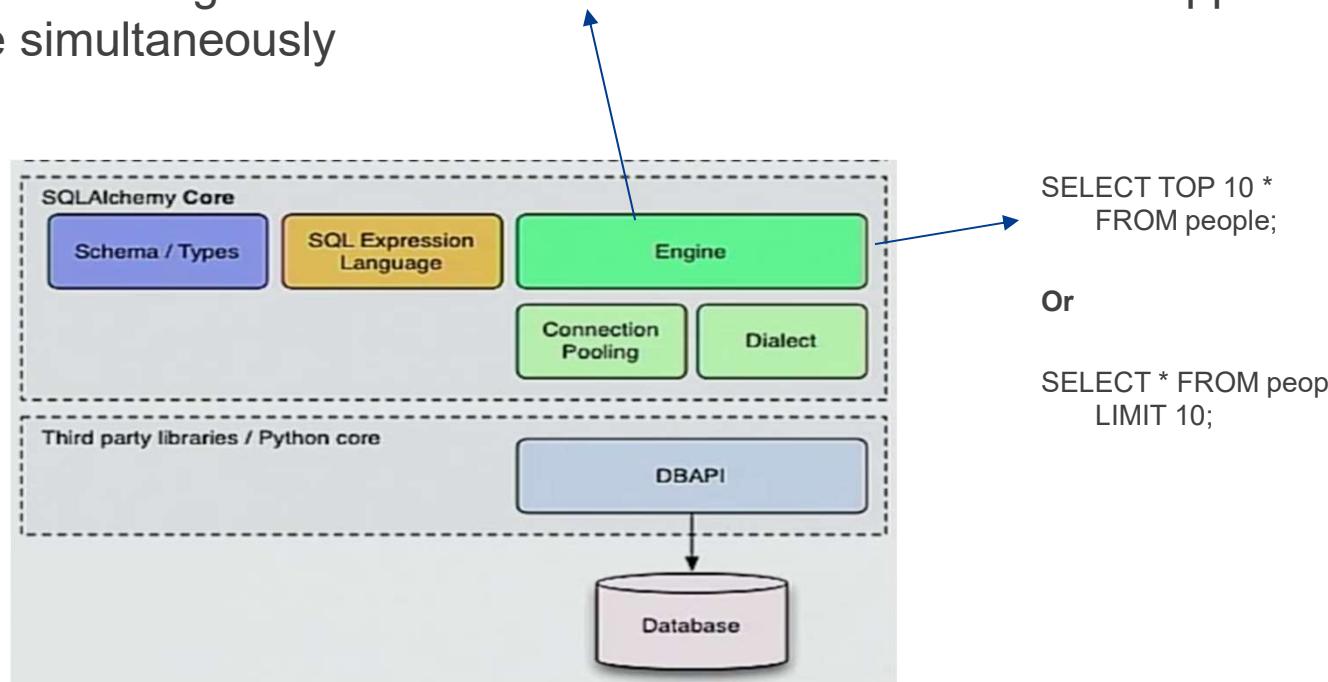
```
from sqlalchemy import create_engine
engine = create_engine('postgresql://usr:pass@localhost:5432/sqlalchemy')
...
engine = create_engine('sqlite:///some.db')
```



# Directly Executing SQL

Connection Pooling ( $\approx$  connection sharing)

- » Creating DB connections are expensive
- » With pooling, program fetches an existing connection, use and put it back into pool
- » easier management of the number of connections that an application might use simultaneously



# Directly Executing SQL (SQLAlchemy Core)

## SQL Expression Language

```
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))
)
```

```
>>> 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
()
```

80 <http://docs.sqlalchemy.org/en/latest/core/tutorial.html>

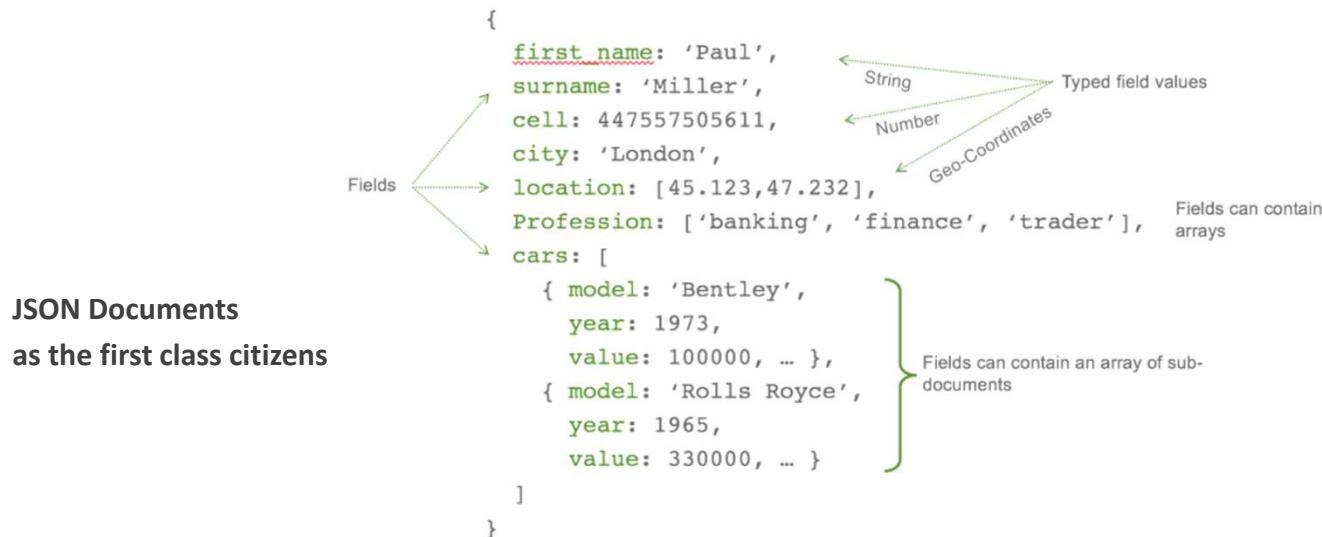
# MongoDB Query

The basic idea of databased connectivity API applies with MongoDB too ...

Many implementations

Direct: PyMongo, Motor

ORM-like: PyMODM, MongoEngine, etc.



<https://www.mongodb.com/blog/post/getting-started-with-python-and-mongodb>

# MongoDB Query

MongoDB Server (download, install, run ...) and MongoDB Client (connect, create db, ...)

```
db.users.insertOne(           ← collection → Create one if the collection doesn't exist
{
    name: "sue",             ← field: value
    age: 26,                 ← field: value
    status: "pending"        ← field: value
}
)
db.users.find(
    { age: { $gt: 18 } },
    { name: 1, address: 1 }
).limit(5)                   ← collection
                             ← query criteria
                             ← projection
                             ← cursor modifier

db.users.updateMany(
    { age: { $lt: 18 } },     ← collection
    { $set: { status: "reject" } } ← update filter
)
db.users.deleteMany(
    { status: "reject" }      ← collection
)
                           ← delete filter
```

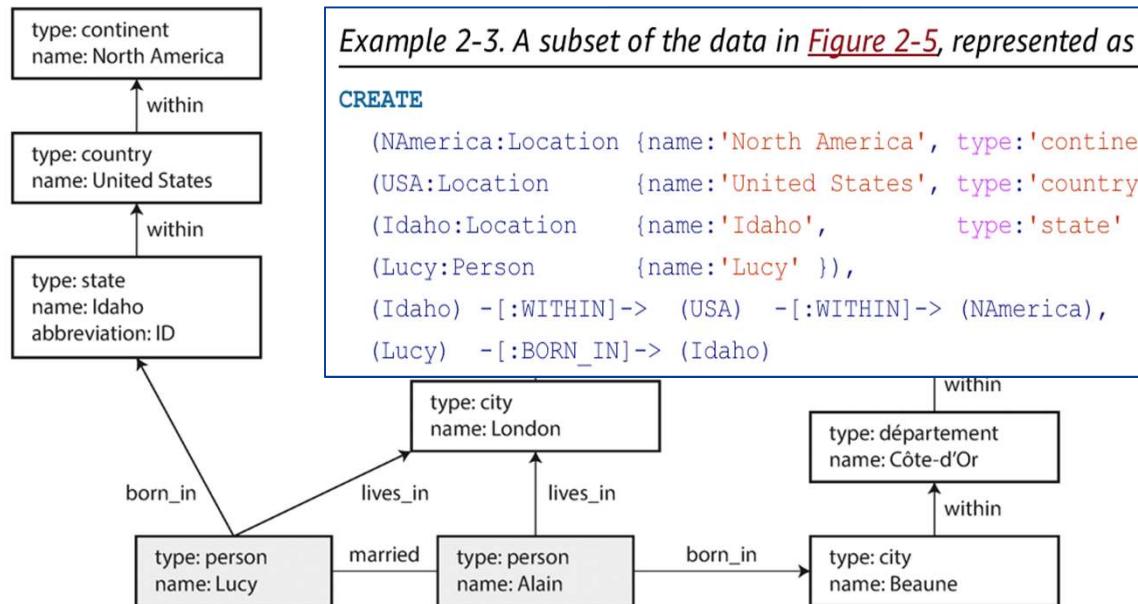
← collection  
← field: value  
} document  
← field: value  
← field: value  
← collection  
← query criteria  
← projection  
← cursor modifier  
← collection  
← update filter  
← update action  
← collection  
← delete filter

<https://docs.mongodb.com/manual/crud/>

# Querying Graph-like Models

Cypher Query - declarative query language for graphs

<https://neo4j.com/developer/example-project/>



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

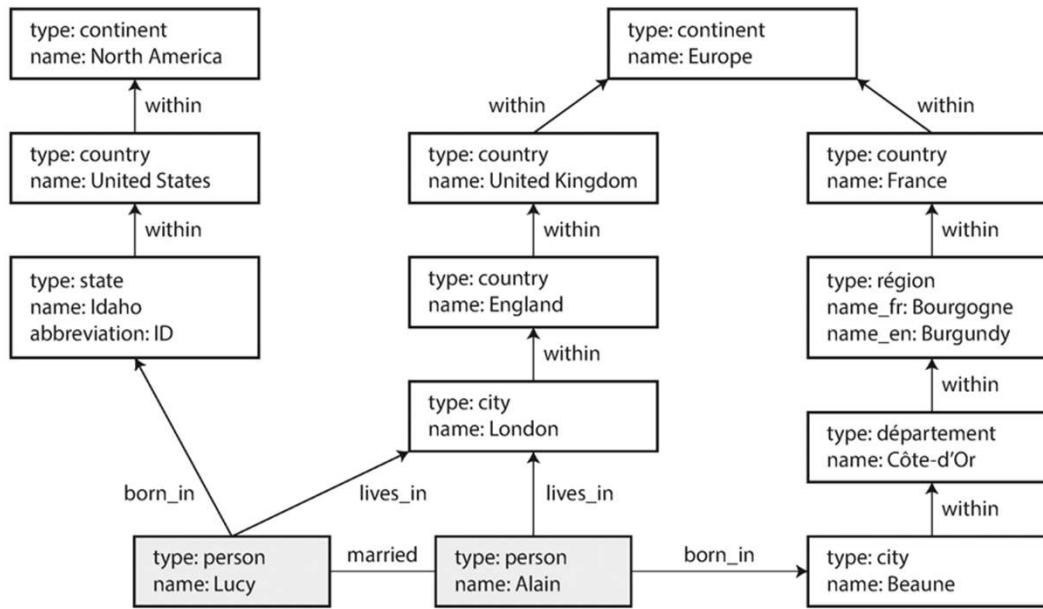
**CREATE**

```
(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*

```

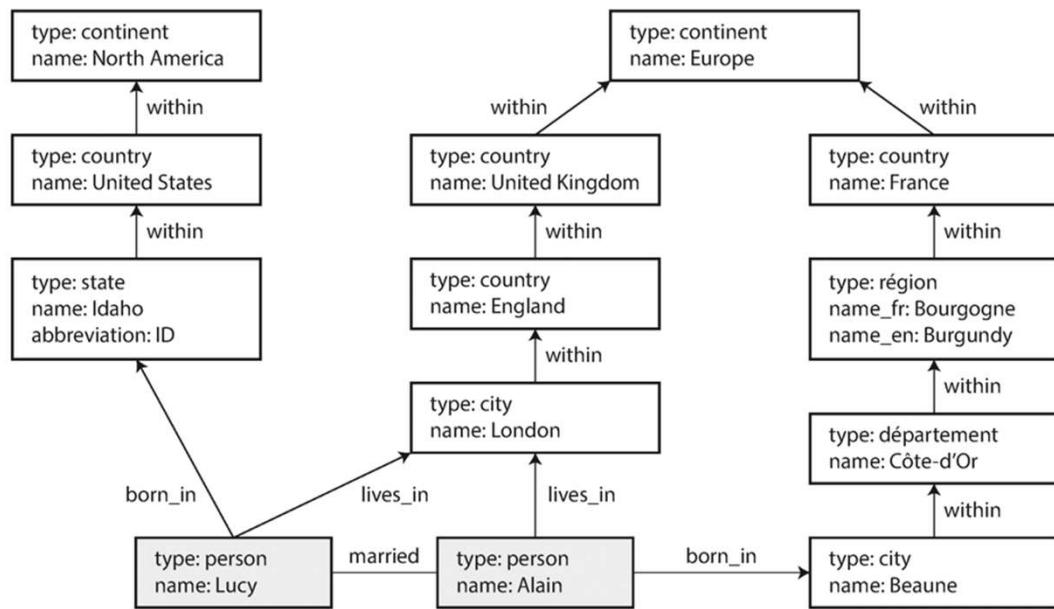
MATCH
    (person) -[:BORN_IN]-> () -[:WITHIN*0..]-> (us:Location {name:'United
    States'}),
    (person) -[:LIVES_IN]-> () -[:WITHIN*0..]-> (eu:Location
    {name:'Europe'})
RETURN person.name
  
```



“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
);

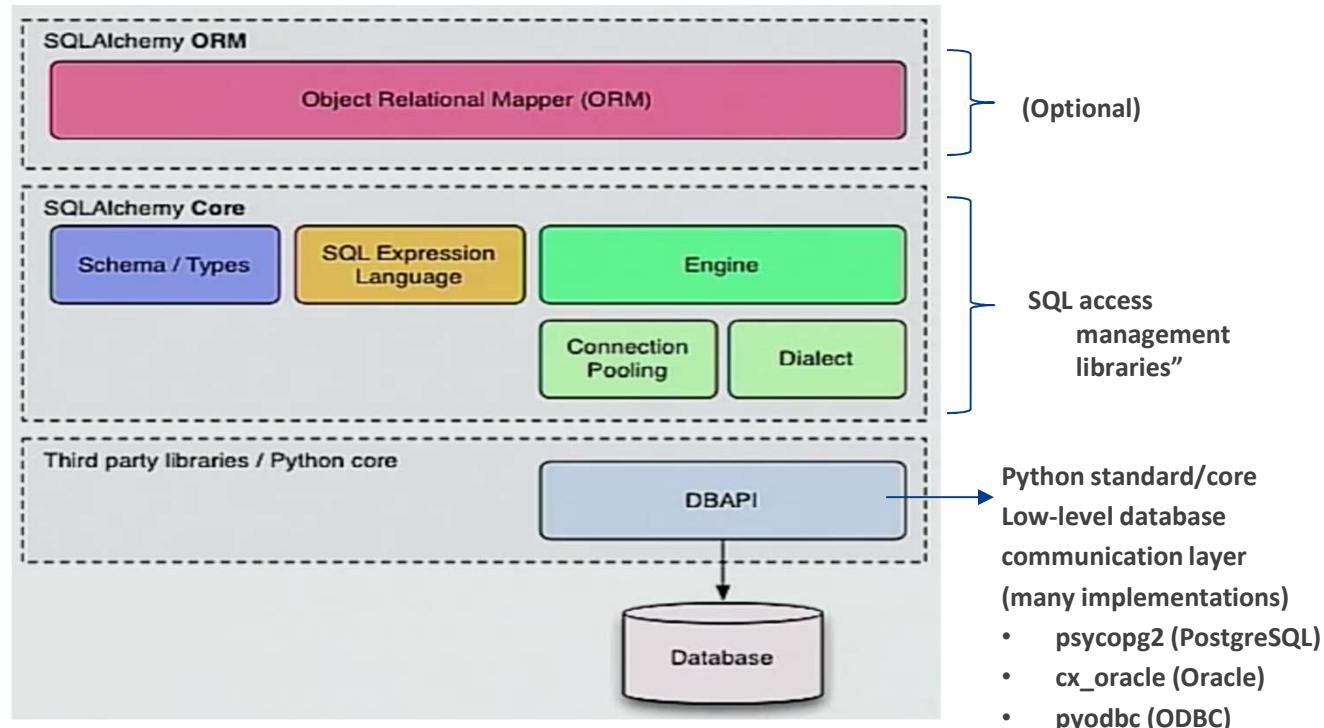
CREATE INDEX edges_tails ON edges (tail_vertex);
CREATE INDEX edges_heads ON edges (head_vertex);
  
```

## 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' ①
    UNION
    SELECT edges.tail_vertex FROM edges ②
        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' ③
    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 (④
    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 (⑤
    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 ⑥
JOIN lives_in_europe ON vertices.vertex_id =
lives_in_europe.vertex_id;
```

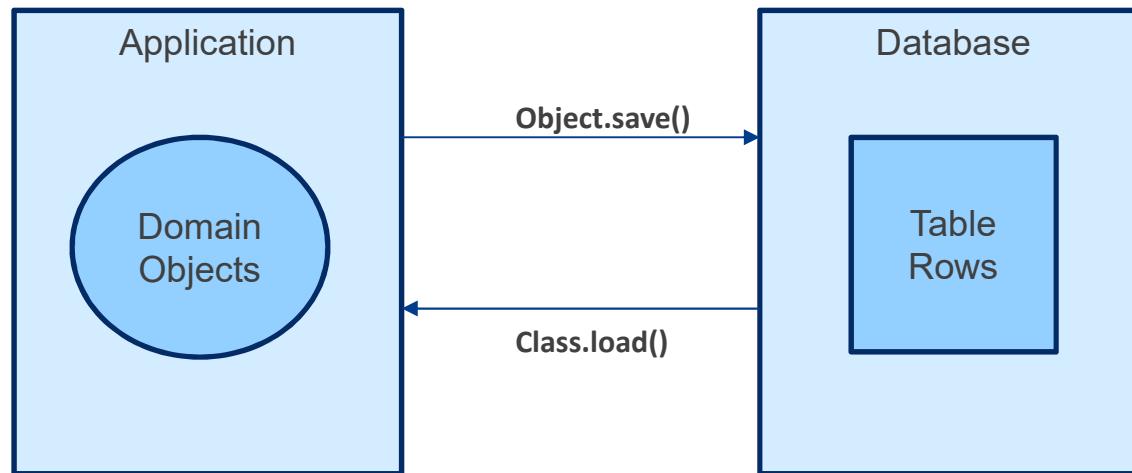
# Object Relational Mapping (ORM)

e.g., Python SQLAlchemy



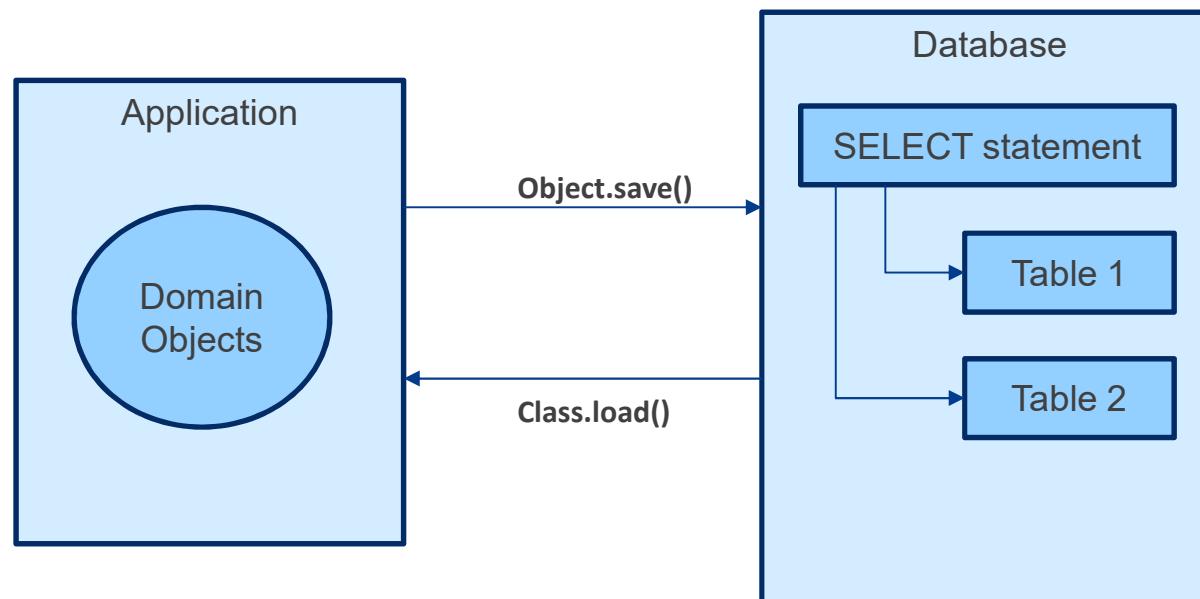
# ORM

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



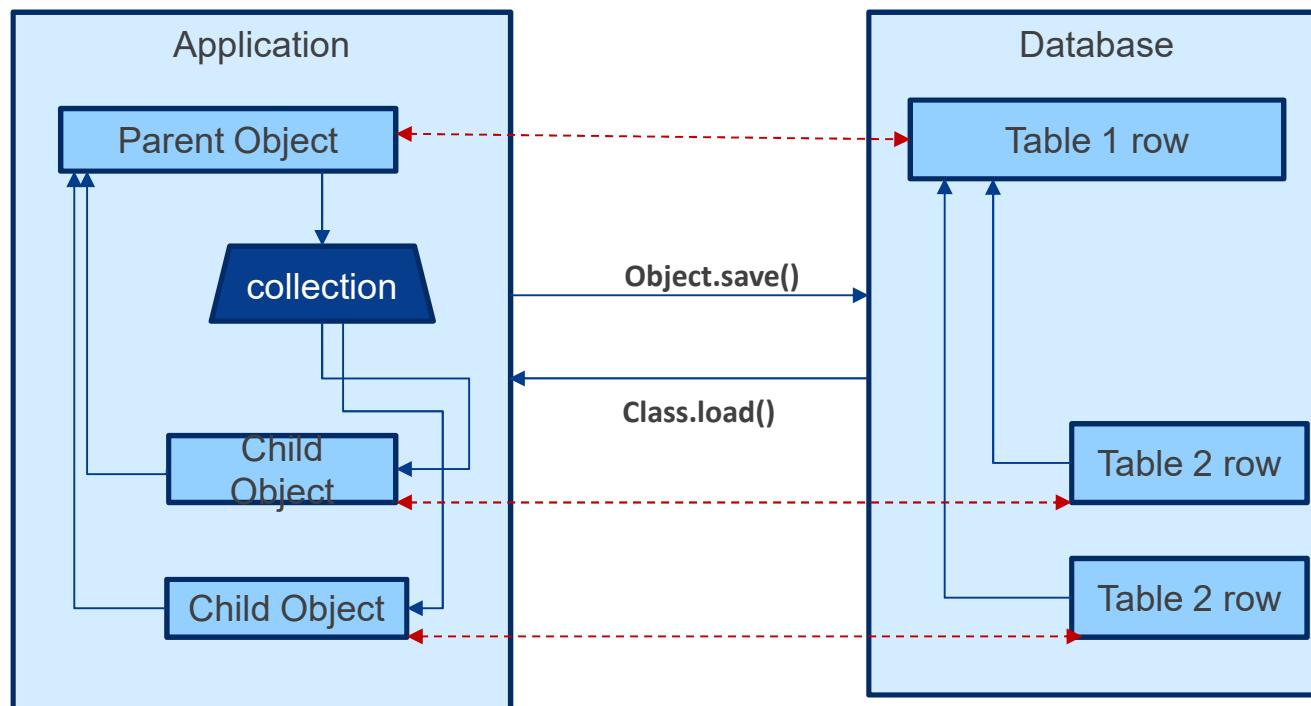
# ORM

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



# ORM

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



# 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)

# 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'

# 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()
```

# 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:

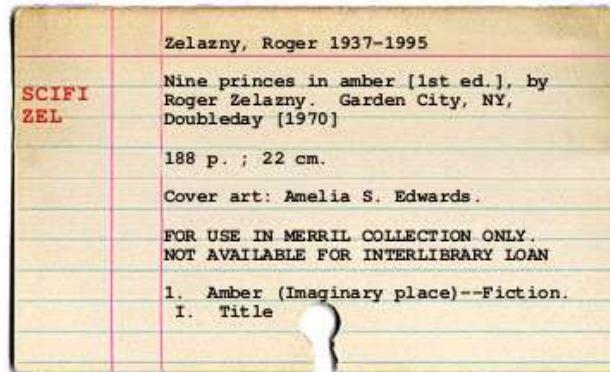
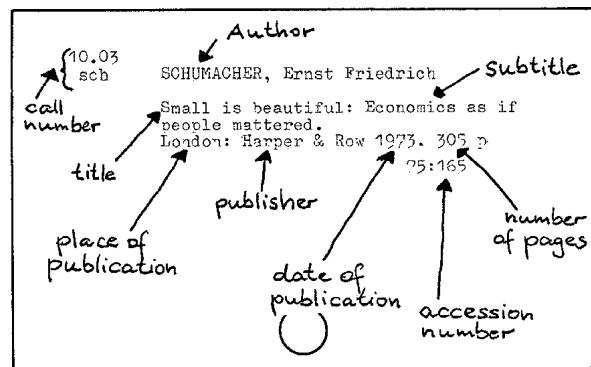
- Phone numbers (caller, recipient)
- Time and duration of the call
- Mobile phone locations (caller, recipient)
  - If mobile phone is in connection with local cell towers, a record of your location at any given moment ...

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.

Metadata (The MIT Press Essential Knowledge Series)

# 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 |
|--------------------------|----------|---------------------|----------------------|-------------|-------|
| <i>Intellectual Pro-</i> | Palfrey, | 2012                | Intellectual proper- | HD53 .P35   | 172   |

# Managing and Publishing Metadata

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

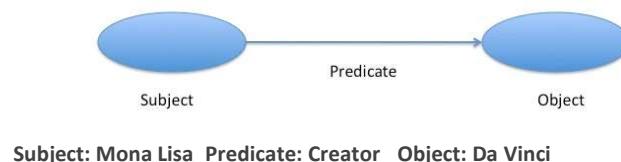
Metadata is a “map”, is a means by which the complexity of an object is represented in a simpler form

- 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 ...

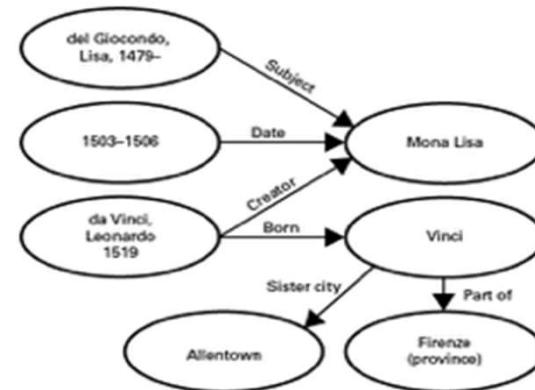
Metadata is a statement about a potentially informative “thing” (resource).

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



Subject 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 ...!!)



# Descriptive Metadata

Standard, the simplest form of metadata: Dublin Core

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

## 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.

Title: Mona Lisa

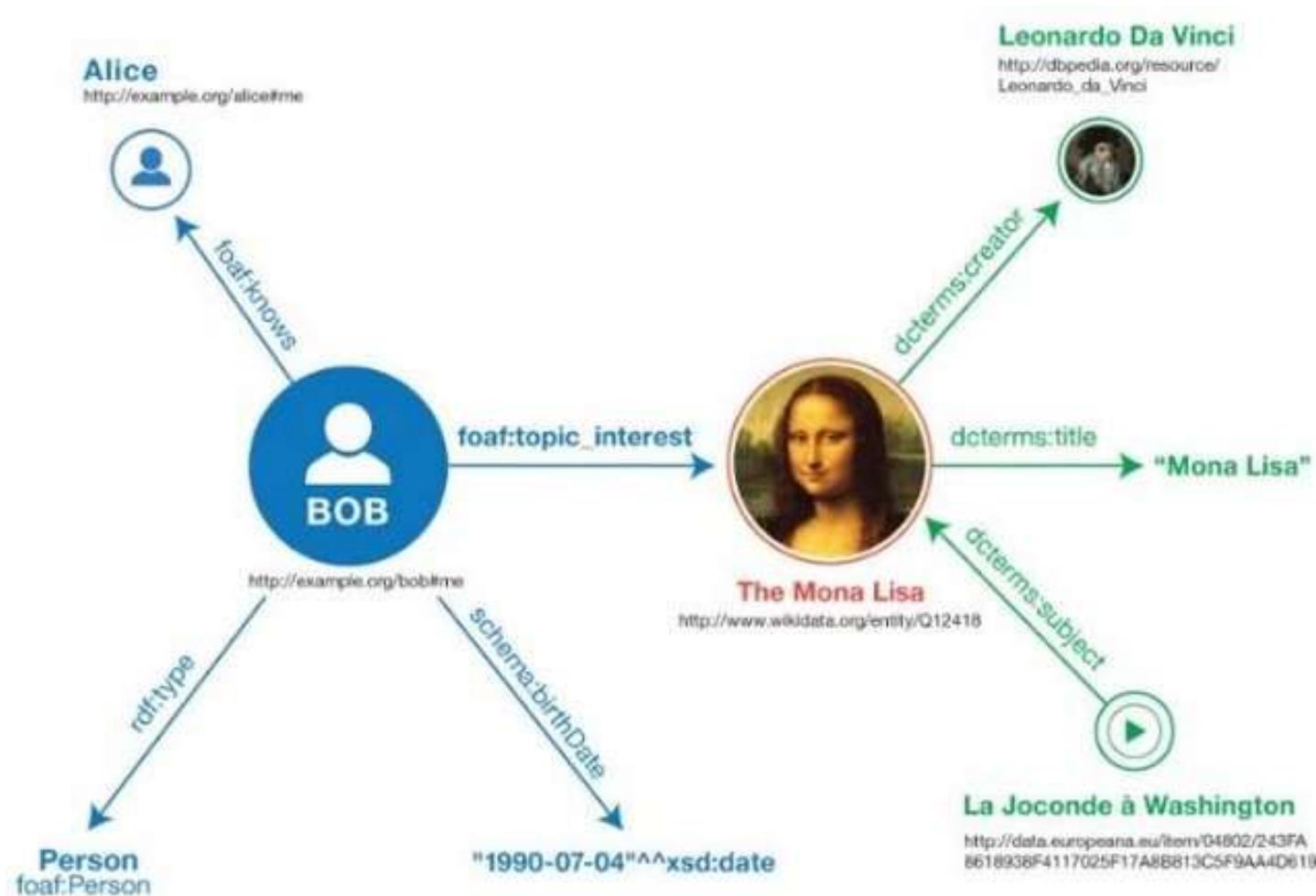
Title: La Gioconda

Creator: Leonardo da Vinci

Subject: Lisa Gherardini

Date: 1503–1506

# Descriptive Metadata



# RDF Example (with Dublin Core)

[https://www.w3schools.com/xml/xml\\_rdf.asp](https://www.w3schools.com/xml/xml_rdf.asp)

10      Metadata (The MIT Press Essential Knowledge Series)  
2