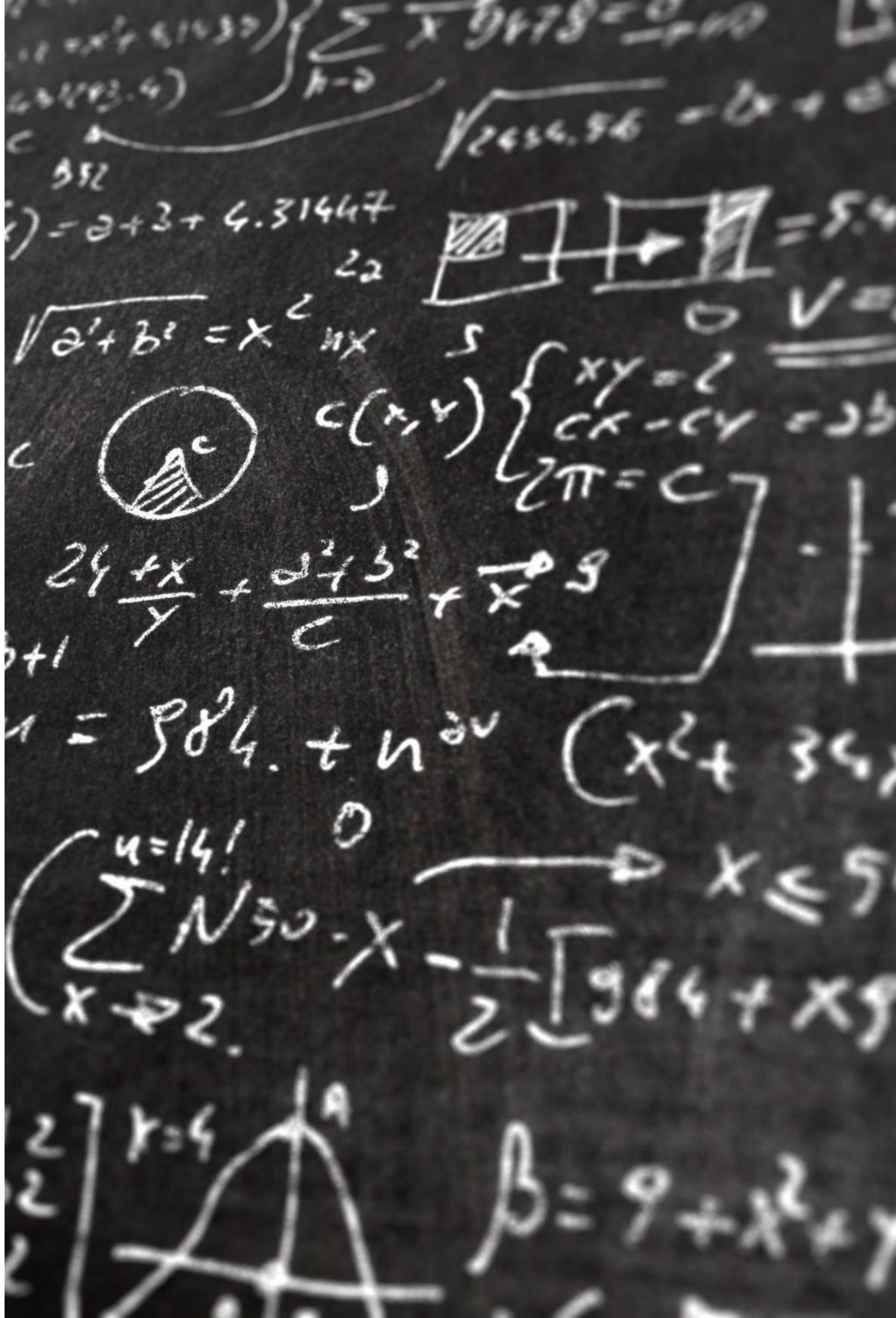
Data models and query languages

Toni Espinosa 17/02/25



Why are data models important?

How software is designed and developed

website users?

dynamic web content?

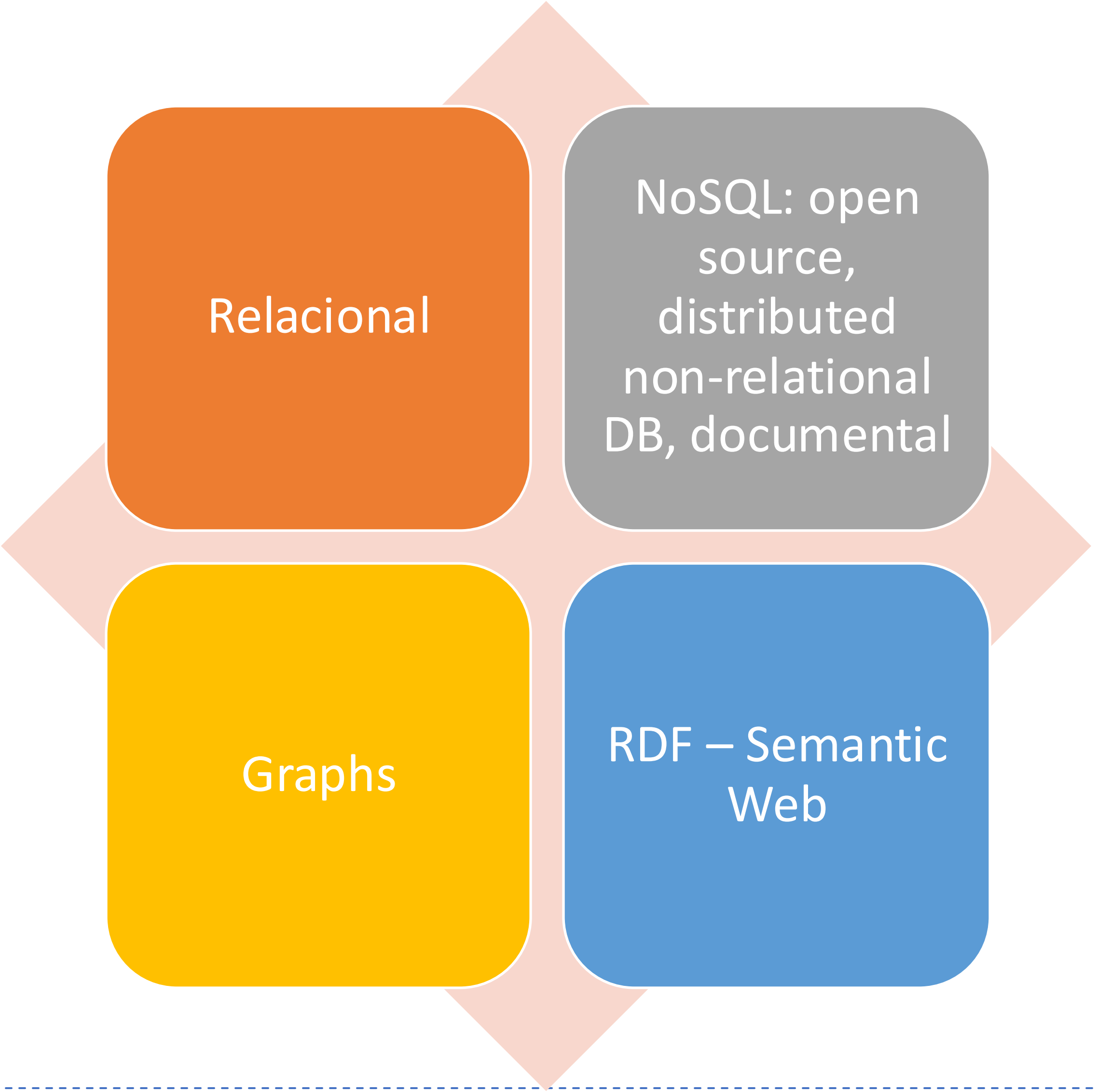
How do we think about the problem to solve?

Applications are operations applied to data -> new data

Data and points of view

• App developer:	Objects, data structures, APIs
• Data model storage:	JSON, XML, relational tables
• Data system internals:	Bytes to memory/disk/cloud
• Hardware:	electric current, magnetic fields, light pulses

Most common data models



Relacional

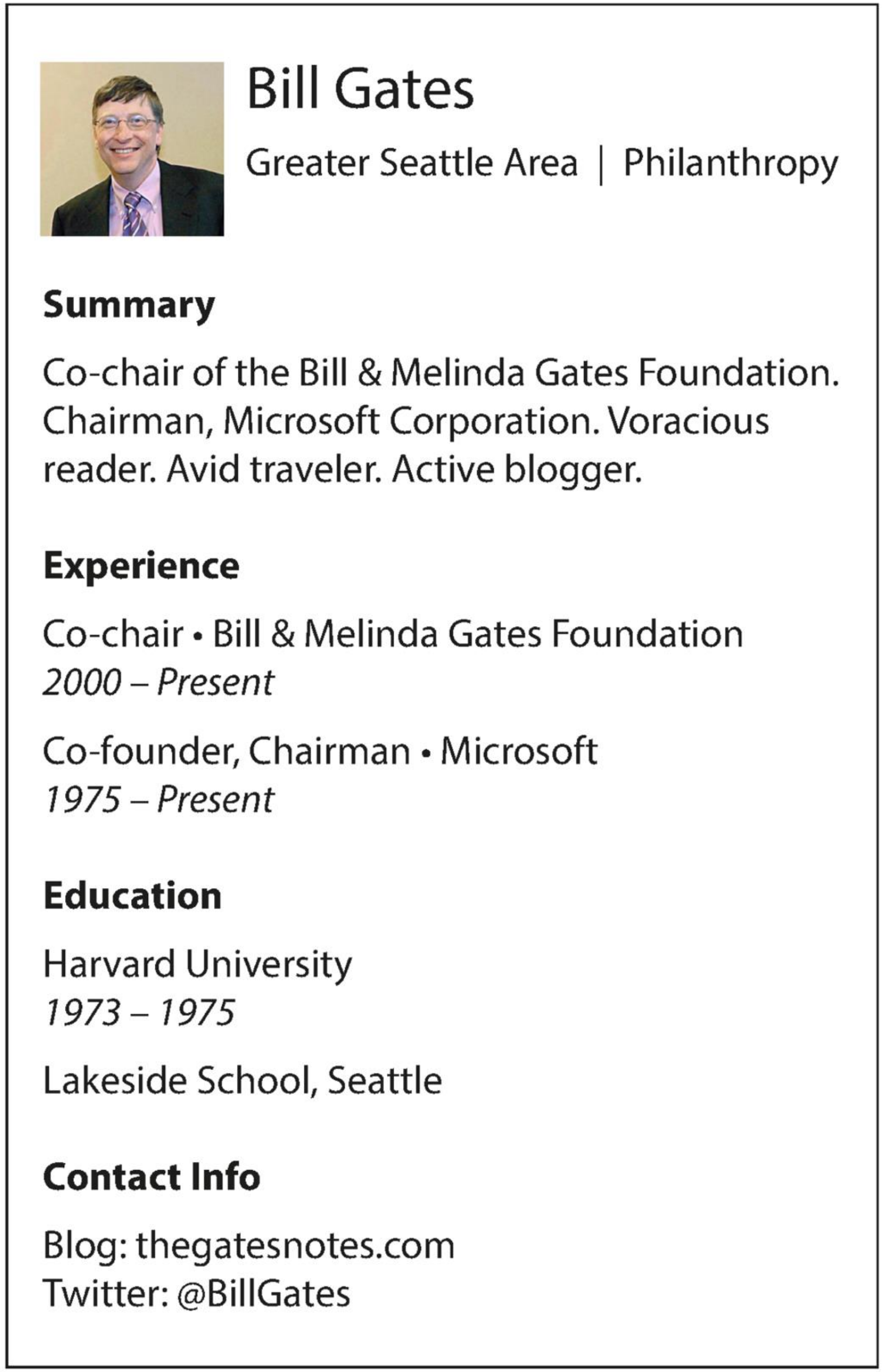
NoSQL: open source, distributed non-relational DB, documental

Graphs

RDF – Semantic Web

How to store a Linkedin profile?

Do you see data entities?



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

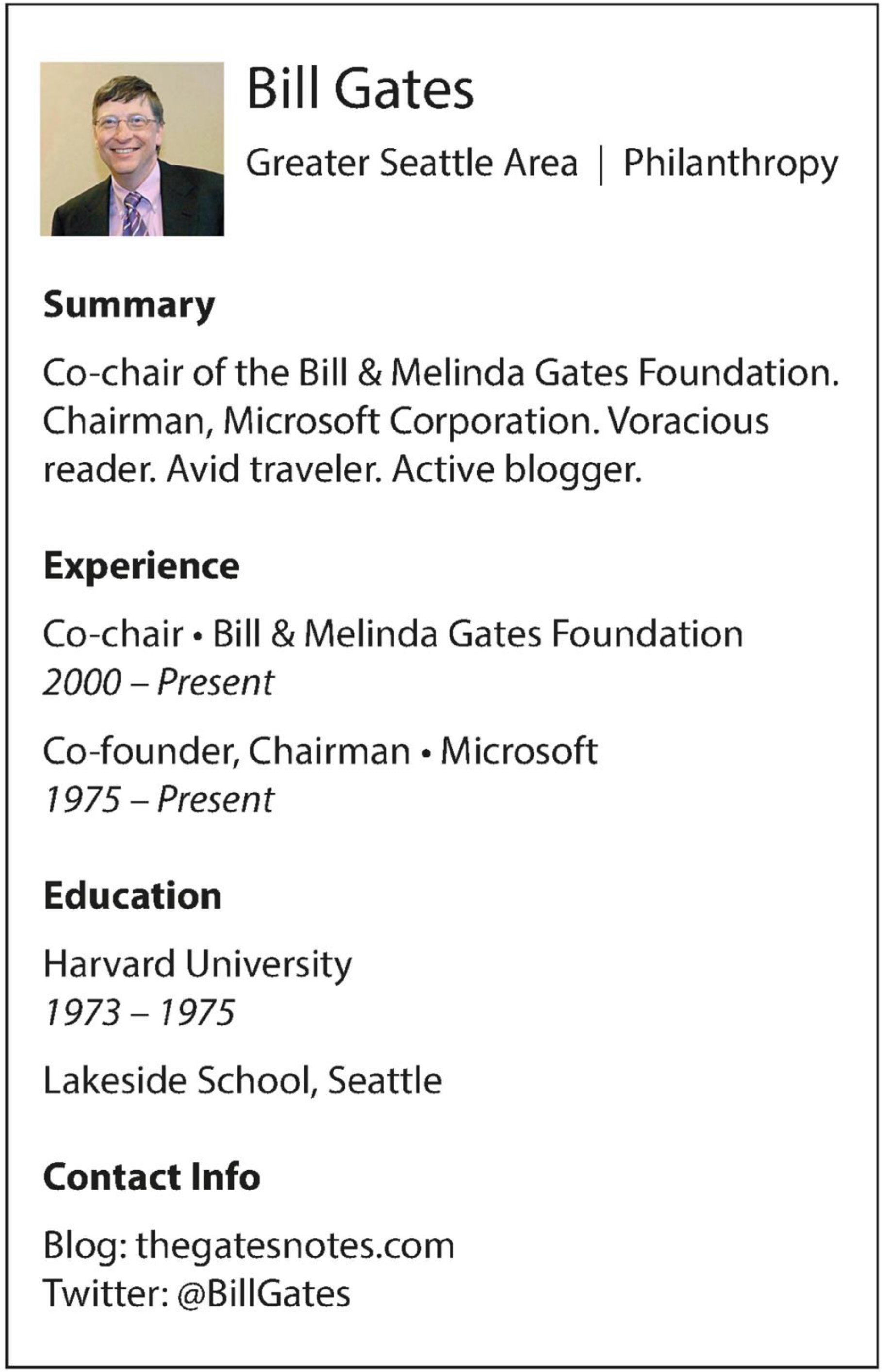
Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

How to store a Linkedin profile?

Do you see data entities?

- users
- regions
- industries
- positions
- education
- contact info



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

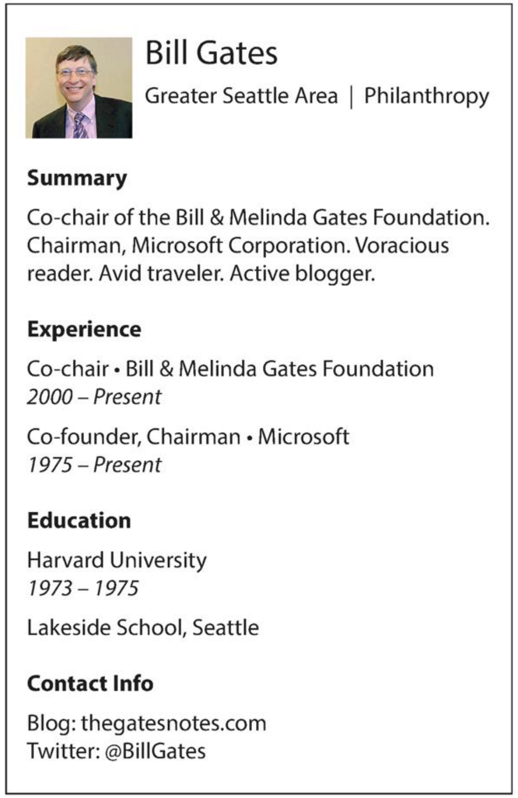
Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

Relational model



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

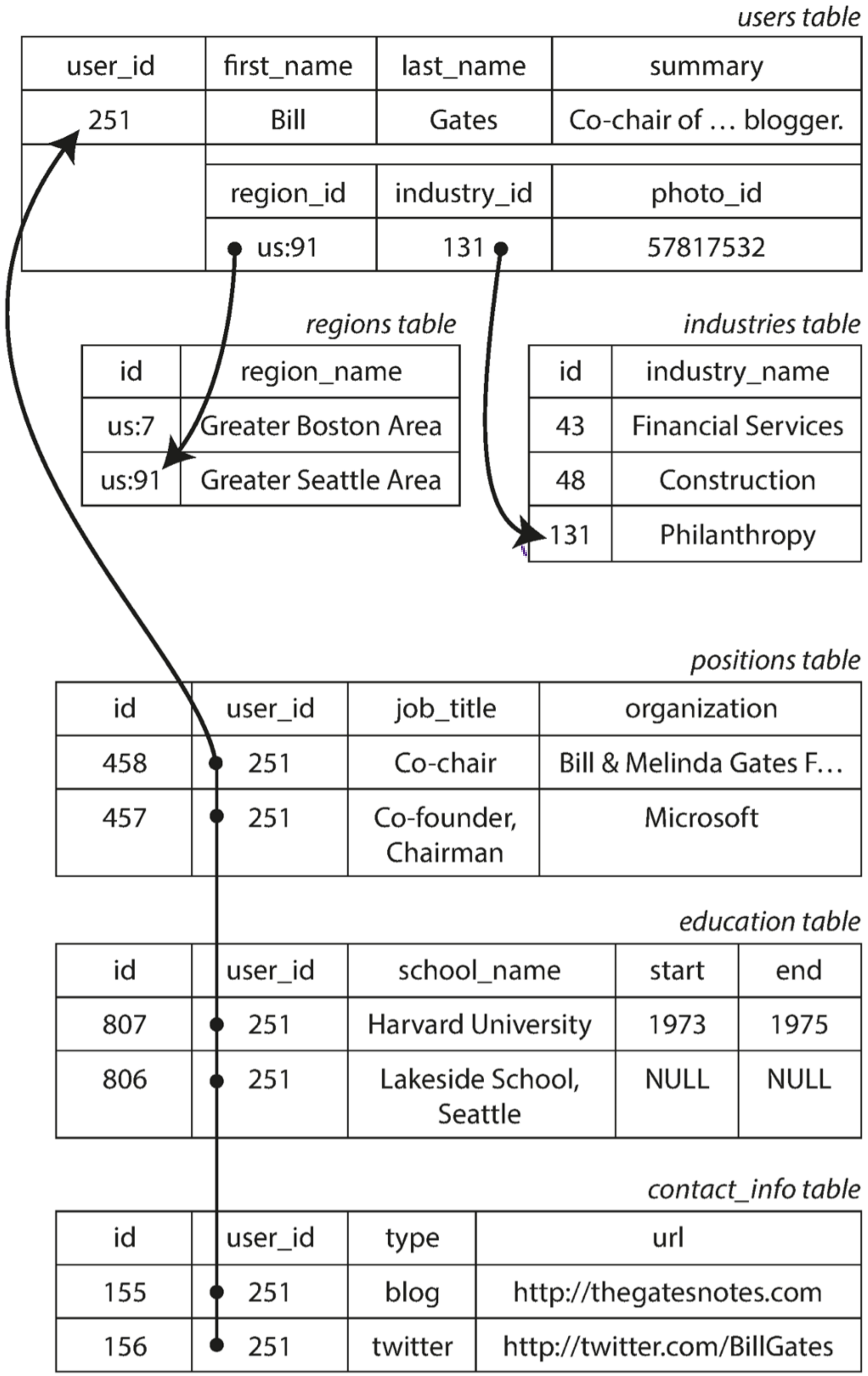
Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates



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	http://thegatesnotes.com
156	251	twitter	http://twitter.com/BillGates

Relational model limitations

Volume	Write operations	License
Handle large volumes of data: beyond disk capacity	Manage high volume of data write operations per second: write throughput	Avoid expensive license cost of complex products: Oracle

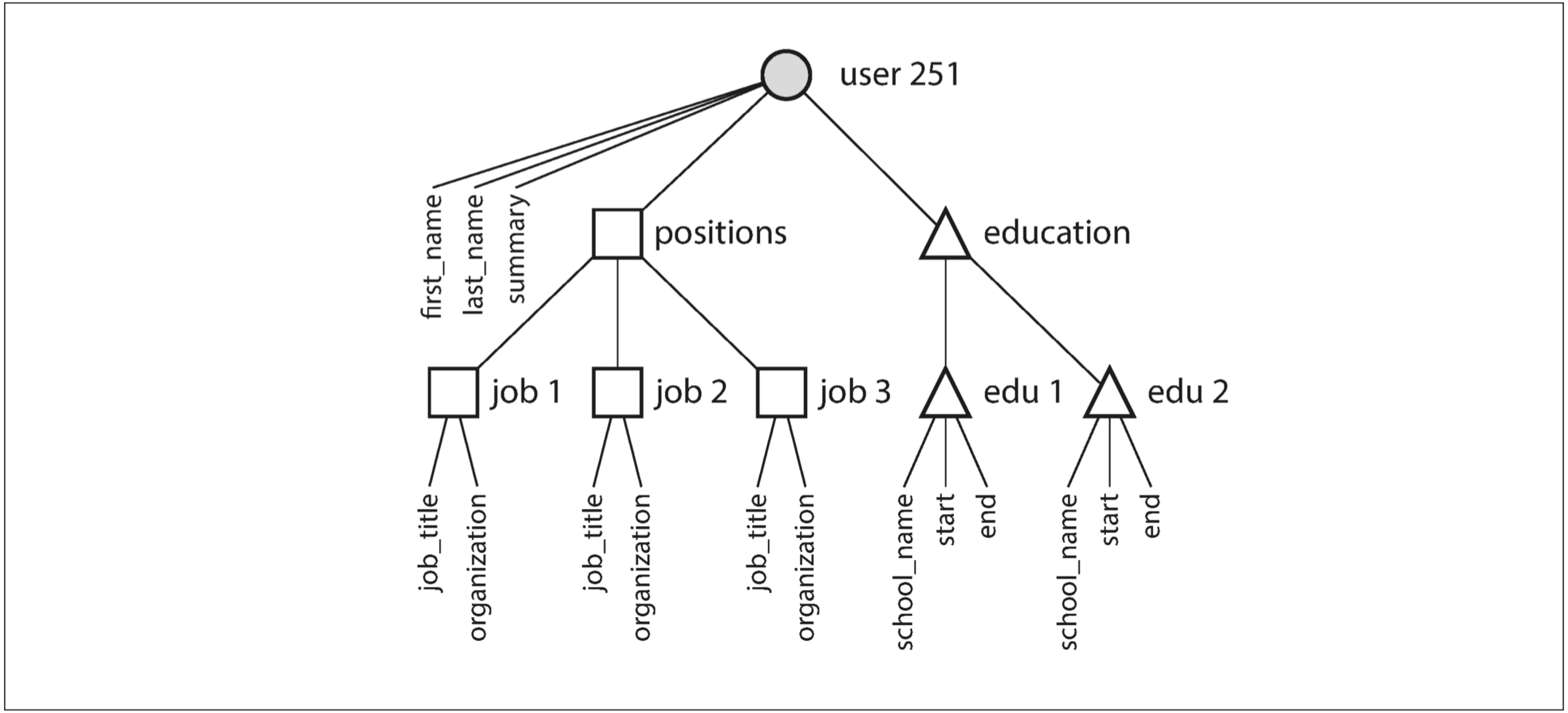
Relational model limitations

- Handle large volumes of data: beyond disk capacity
- Manage high volume of data write operations per second: write throughput
- Avoid expensive license cost of complex products: Oracle

New needs:

- Specific search operations
- Dynamic and expressive data models: easy to change schema

one-to-many relationships from the user profile



JSON Tree representation is explicit

```
{
  "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"
  }
}
```

Need to explore a complete linkedin profile

Mysql DB: **multiple queries** or multi-join between users and other tables

JSON: everything in a single query -> good **data locality**

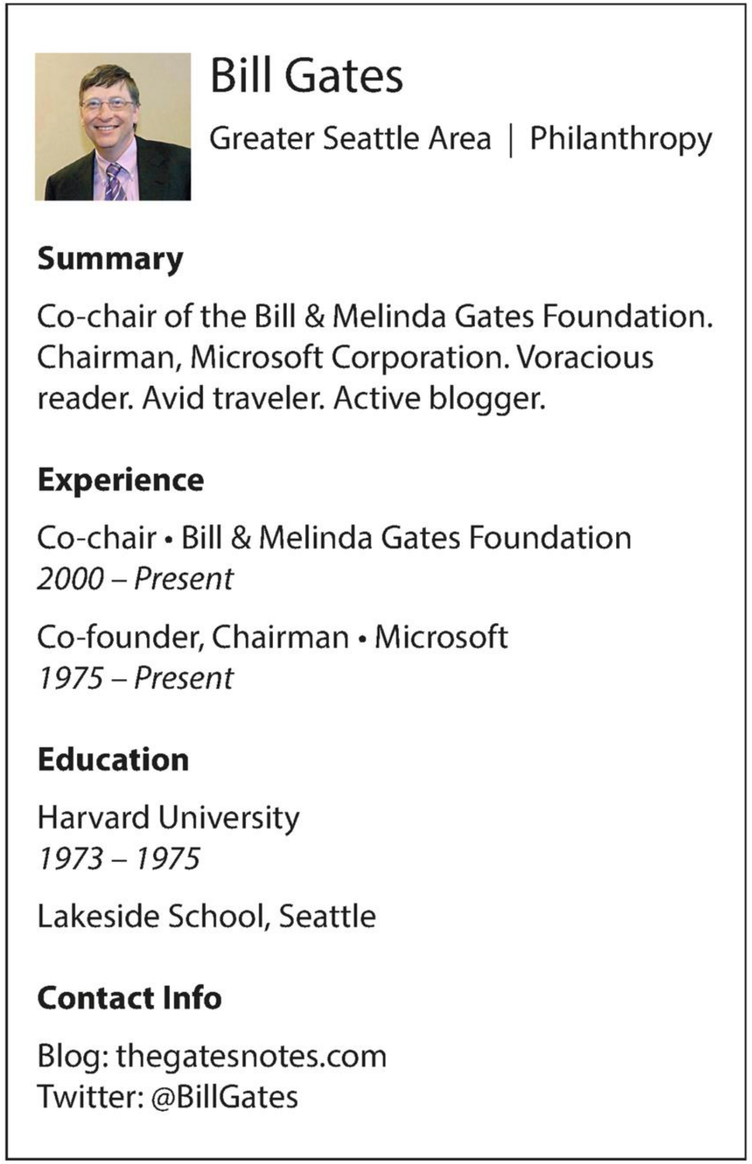
Data locality for queries

- A document is usually stored as a **single continuous string**: encoded as JSON, XML, or plain text
- Need to access different parts of entire document
- If data is in a Single JSON document:
  - faster access because it is **already in memory**
- If data is split across multiple tables:
  - **multiple index lookups** are required to retrieve all data
  - requires more **disk search operations**

Need to explore a complete linkedin profile

But, if we use JSON as a database

how do we manage entity relationships?



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
Twitter: @BillGates

Why do we need numerical identifiers?

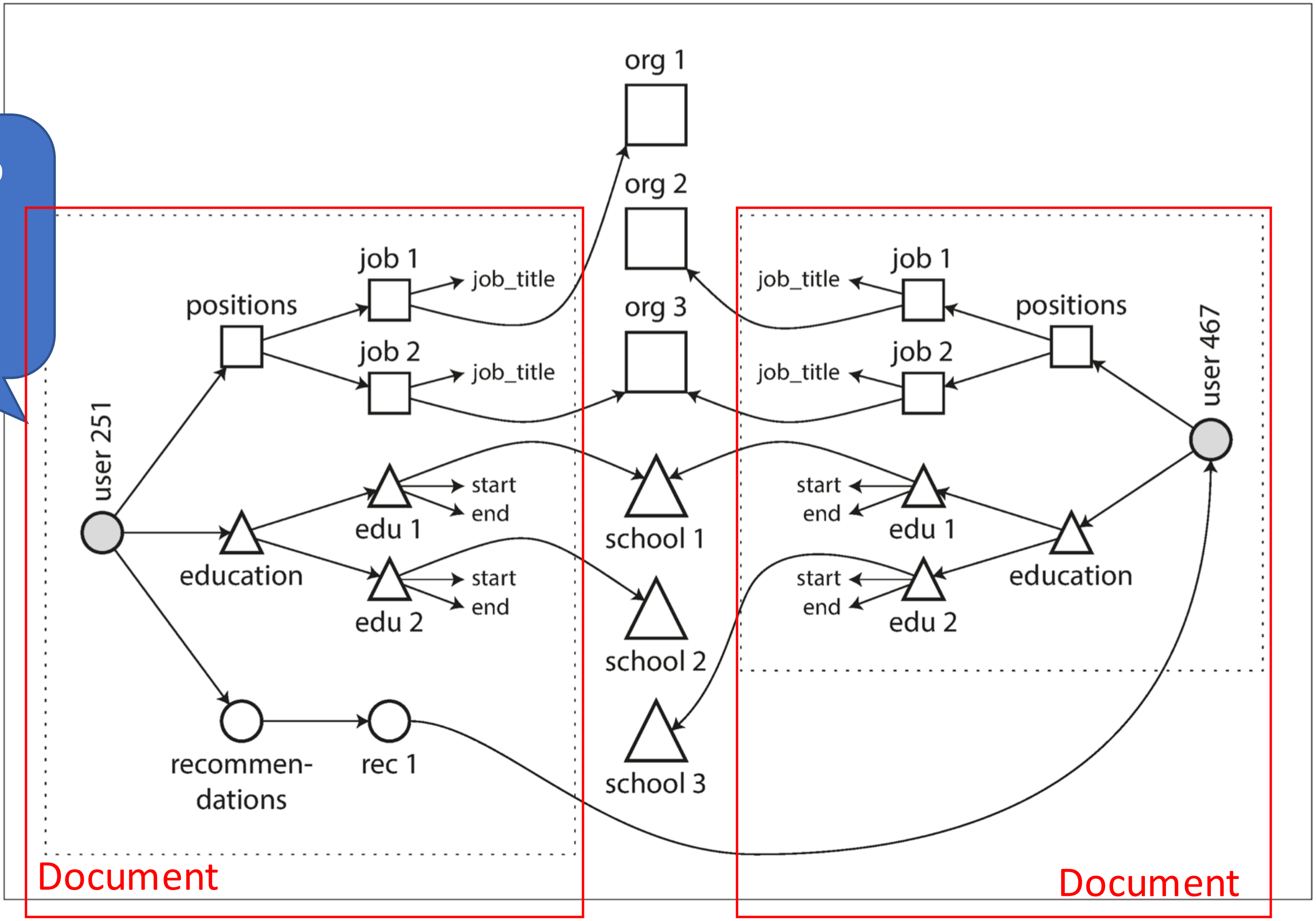
```
{
  "user_id":     251,
  "first_name":  "Bill",
  "last_name":   "Gates",
  "summary":     "Co-chair of the Bill & Melinda Gates... Active blogger.",
  "region_id":   "us:91",
  "industry_id": 131,
  "photo_url":   "/p/7/000/253/05b/308dd6e.jpg",

  "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"
  }
}
```

Entity relationships

region_id and industry_id are identificators. Why?

- Manage **ambiguity**: John Smith
- Easier **updates**: company data can change
- Easier to **translate**: information is isolated
- Easier to **search** for categories than in plain text

organizations and schools as entities



relationship between users and schools?

between users and orgs?

Best data model?

Which data model leads to a simpler application code?

one-to-many / many-to-many queries

Management problems

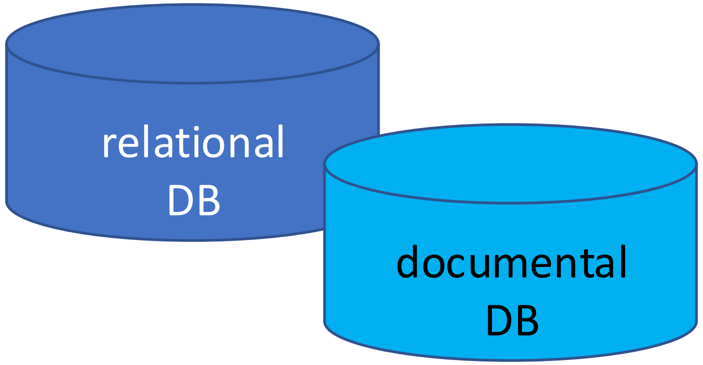
Documental DBs are not very useful for entity join queries

Relational DB join operation is a good tool for these queries

Which is the best model for my application?

Ask yourself: which are the most common operations?

- Data is mostly documents
  - one-to-many relationship tree
  - Load the tree just once
  - Not many reference outside the tree



- Some entities and need to analyse relationships
  - most queries are many-to-many
  - Information structure will not change too much in the future
  - Not a large amount of information changing in time
  - Not thousands of entities

Data change management

Is it possible to modify the data model?

Which is the cost of evolving the models?

Dynamic data schema

- We need to change data format
- Example: separate user name and surname

how do we deal with this change in each data model type?

document (dynamic) / relational (static)

documental DB – dynamic model

- Write new documents with new fields: name and surname
- App will manage difference between model documents

```
if(user && user.name && !user.first_name){
     //old docs do not have surname
     user.first_name = user.name.split(“ “)[0]
}
```

documental DB – dynamic model

- Each change forces a modification of the whole database
- **Migration process**

```
ALTER TABLE users ADD COLUMN first_name text;
UPDATE users SET first_name =
    substring_index(name, ‘ ‘, 1);
```

Pros and Cons

- UPDATE has to **modify each element** of a table or more
- Schema modifications can be **very slow**

- Many times data is heterogeneous
  - Many kinds of objects make unpractical to have a table per object
  - Data structure is **determined by an external entity** that introduces changes in every new version: REST API

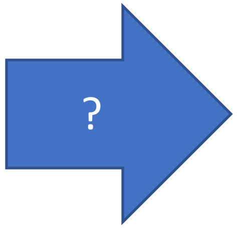
schema-on-read / schema-on-write

- **schema on read**: Data structure is interpreted when reading
  - Data structure is implicit
  - Similar to Python’s dynamic types
  - Maximum flexibility

- **schema on write**: all stored data must follow a pre-defined model
  - Explicit data schema
  - Similar to C++ static types
  - Solid structure

Link data types and schema

1. JSON google maps
2. BBVA bank account
3. UAB registered students
4. Twitter message
5. Education ministry fellowships

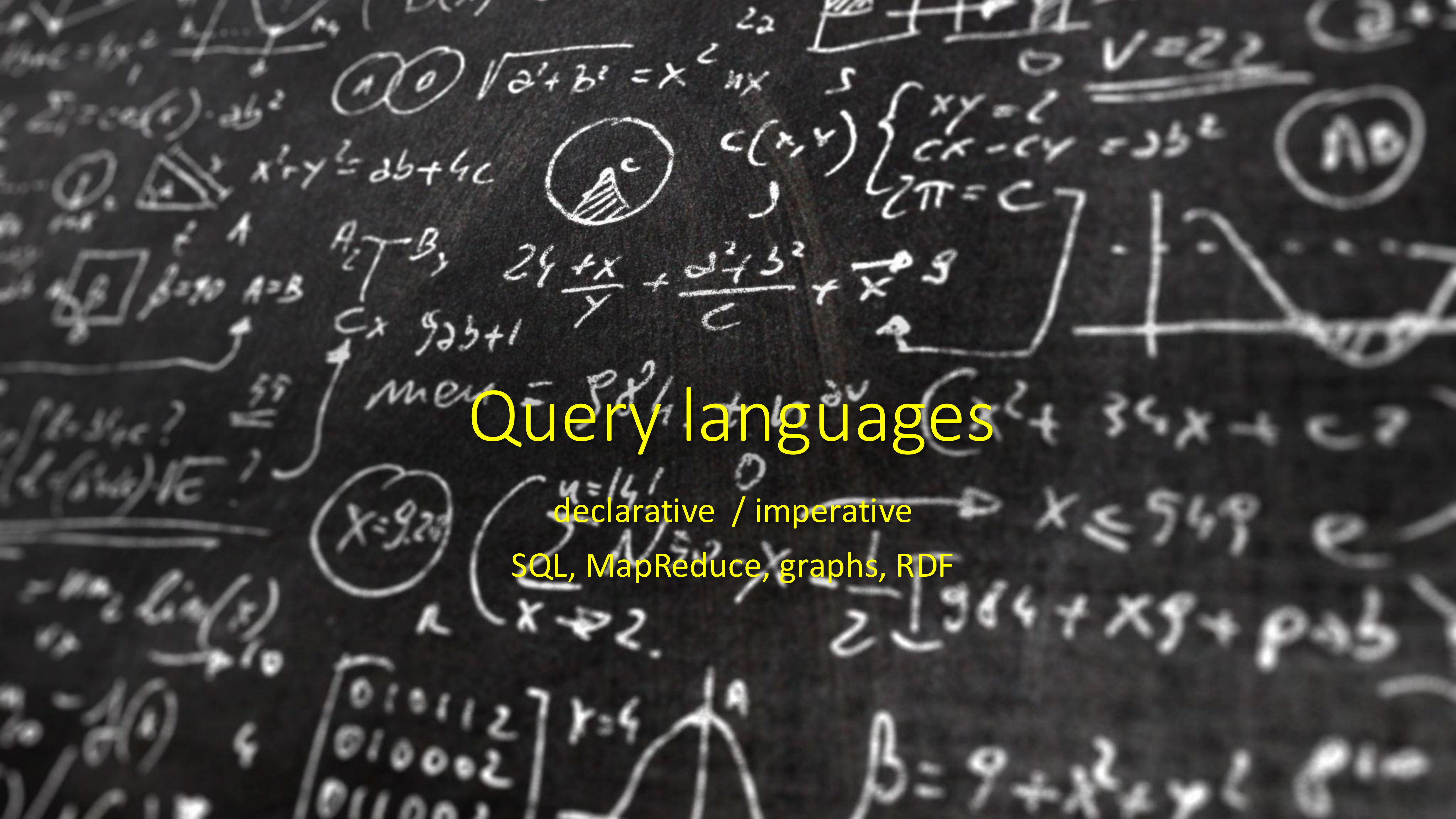


A. Schema on read

B. Schema on write

New DB and hybrid functionality

PostgreSQL, MySQL i DB2 support XML and JSON documents	search, index, modification of files as they were tables
RethinkDB	allows document join operations
MongoDB	client join operations



Query languages

declarative / imperative

SQL, MapReduce, graphs, RDF

Query language

```
function getDolphins(){
  var dolphins=[];
  for(var i=0;i<animals.length;i++){
    if(animals[i].family===“dolphin”){
      dolphins.push(animals[i]);
    }
  }
  return dolphins;
}
```

```
SELECT *
FROM animals
WHERE family=‘dolphin’;
```

which is best?

imperative

```
function getDolphins(){
  var dolphins=[];
  for(var i=0;i<animals.length;i++){
    if(animals[i].family===“dolphin”){
      dolphins.push(animals[i]);
    }
  }
  return dolphins;
}
```

Data structure OPERATIONS and
THEIR ORDER

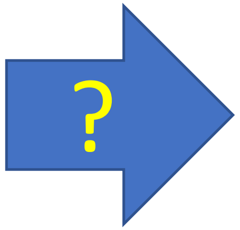
declarative

```
SELECT *
FROM animals
WHERE family=‘dolphin’;
```

DATA PATTERNS
DATA TRANSFORMATIONS

CHOOSE CHARACTERISTICS

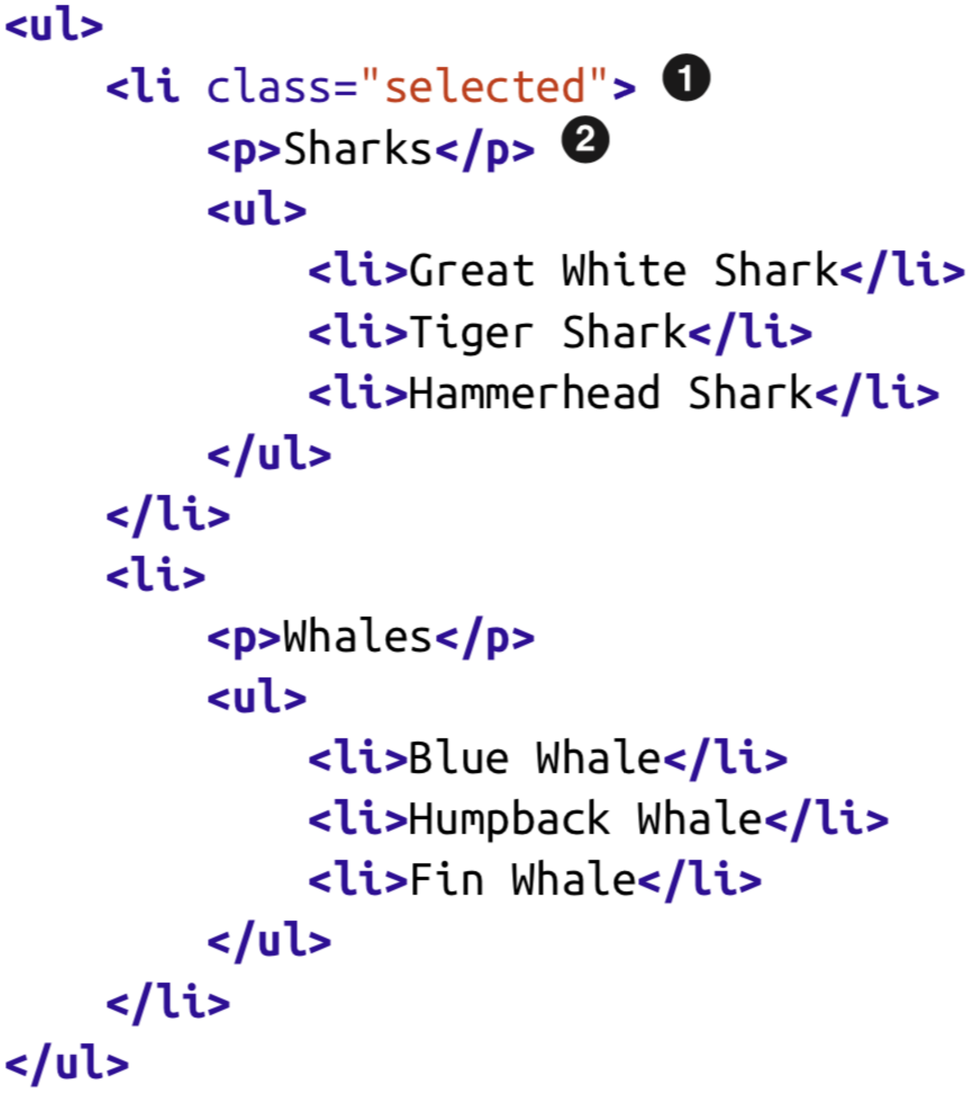
1. Hide implementation details
2. Define an animal order
3. Does not need any order in the operations to do
4. Define a strict order of operations, not easy to make it parallel



declarative (SQL)

imperative (Python)

CSS Declarative Query



```
<ul>
    <li class="selected"> ❶
        <p>Sharks</p> ❷
        <ul>
            <li>Great White Shark</li>
            <li>Tiger Shark</li>
            <li>Hammerhead Shark</li>
        </ul>
    </li>
    <li>
        <p>Whales</p>
        <ul>
            <li>Blue Whale</li>
            <li>Humpback Whale</li>
            <li>Fin Whale</li>
        </ul>
    </li>
</ul>
```

1. selected item marked with CSS class = “selected”

2. <p>Sharks</p> is the title of the page

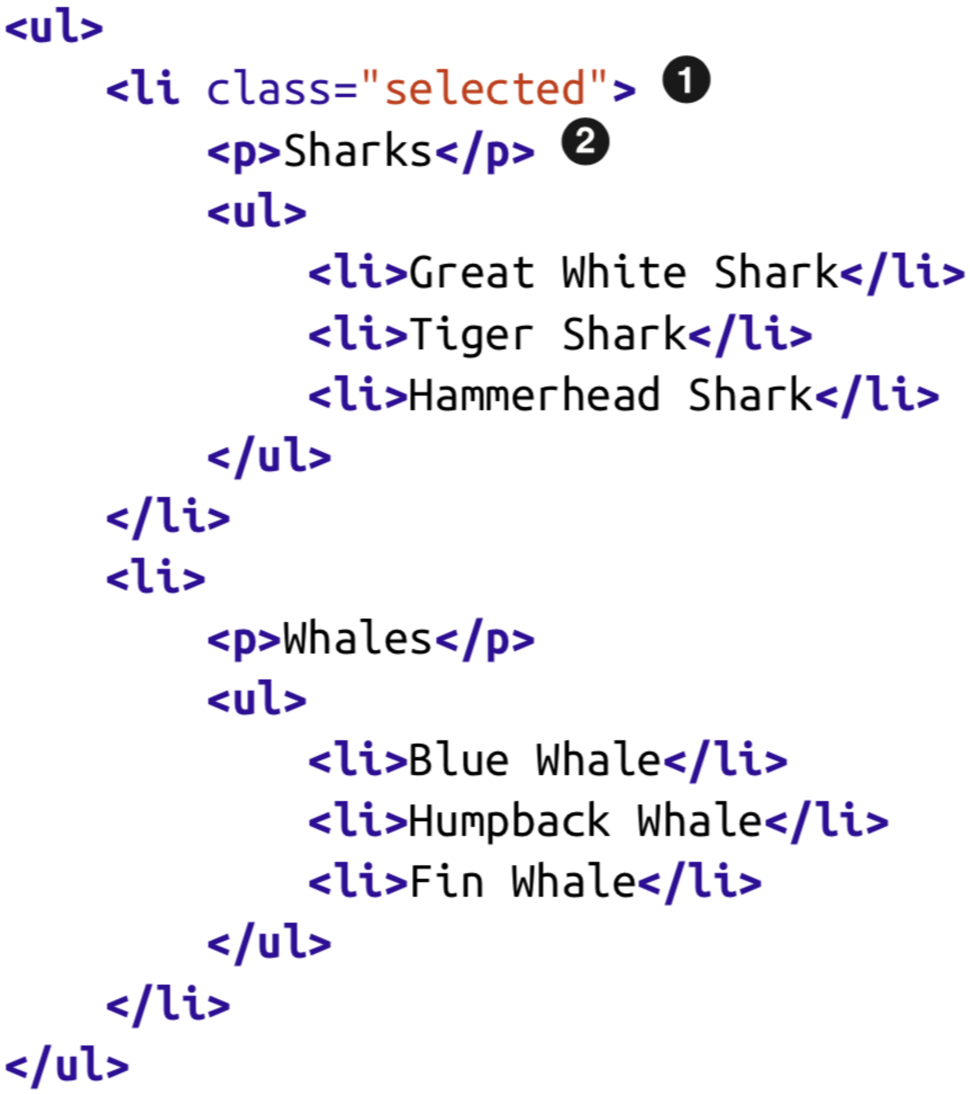
Page selected is blue

```
li.selected > p {
   background-color: blue;
}
```

Declarative language:

which part of the document do we apply the property (blue)

Selected page in blue?



```html
<ul>
    <li class="selected"> ❶
        <p>Sharks</p> ❷
        <ul>
            <li>Great White Shark</li>
            <li>Tiger Shark</li>
            <li>Hammerhead Shark</li>
        </ul>
    </li>
    <li>
        <p>Whales</p>
        <ul>
            <li>Blue Whale</li>
            <li>Humpback Whale</li>
            <li>Fin Whale</li>
        </ul>
    </li>
</ul>
```

```css
li.selected > p {
   background-color: blue;
}
```

li.selected > p

element patterns to apply blue style:

elements <p> with parent <li> with CSS class **selected**

same example with XSL format

li[@class='selected']/p

same to CSS selector:

li.selected > p

```
<xsl:template match="li[@class='selected']/p">
        <fo:block background-color="blue">
                <xsl:apply-templates/>
        </fo:block>
</xsl:template>
```

Managing styles with javaScript (DOM)

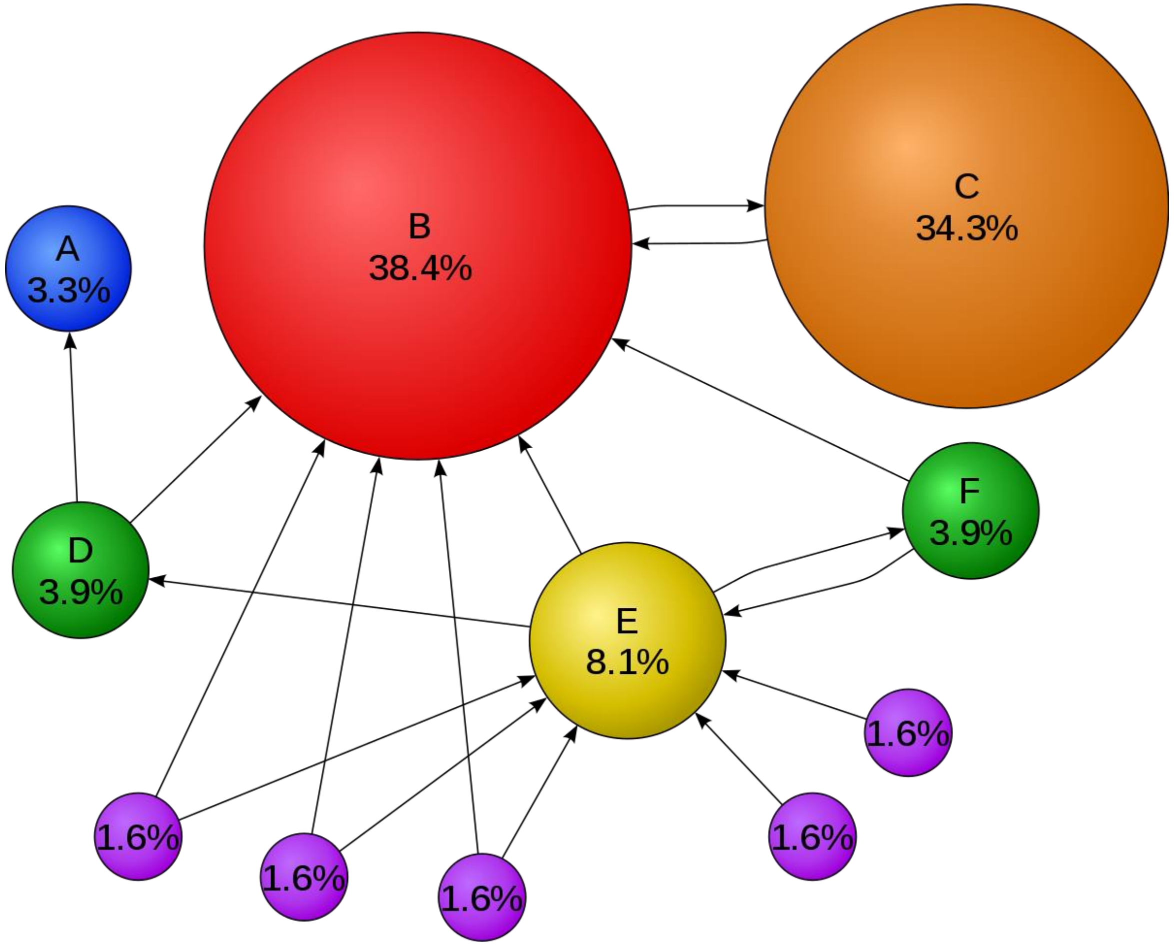
```
var liElements = document.getElementsByTagName("li");
for (var i = 0; i < liElements.length; i++) {
       if (liElements[i].className === "selected") {
             var children = liElements[i].childNodes;
             for(var j = 0; j < children.length; j++) {
                var child = children[j];

                 if(child.nodeType === Node.ELEMENT_NODE&&child.tagName === "P")
                 {
                    child.setAttribute("style", "background-color: blue");

                 }
             }
       }
}
```

Which is the best option?

- CSS (declarative)

- javascript with DOM (imperative)

Graph data models



Graph data models

one-to-many relationships: document model could be used

many-to-many relationships in our data

- Simple cases can be solved with relational models
- Complex connection patterns: graph models

Graph: nodes and links

Social graphs:

- Nodes are people, links are relationships (who knows who)

Web graph

- Nodes are pages connected with html hyperlinks

Map graph

- Nodes are street crossings, links are streets between them

Graph algorithms

Google PageRank

Vehicle navigation systems

Social networks: value of users in terms of their connectivity

- Nodes: people, places, actions, events, comments
- Links: frienships, who is commenting, actions related to places, who attended events, ...

Data models and programming languages

Models

- Property graph: Neo4j, Titan, InfiniteGraf
- Triple-Store: Datomic, AllegroGraph

Declarative languages for graphs

- Cypher, SPARQL, Datalog

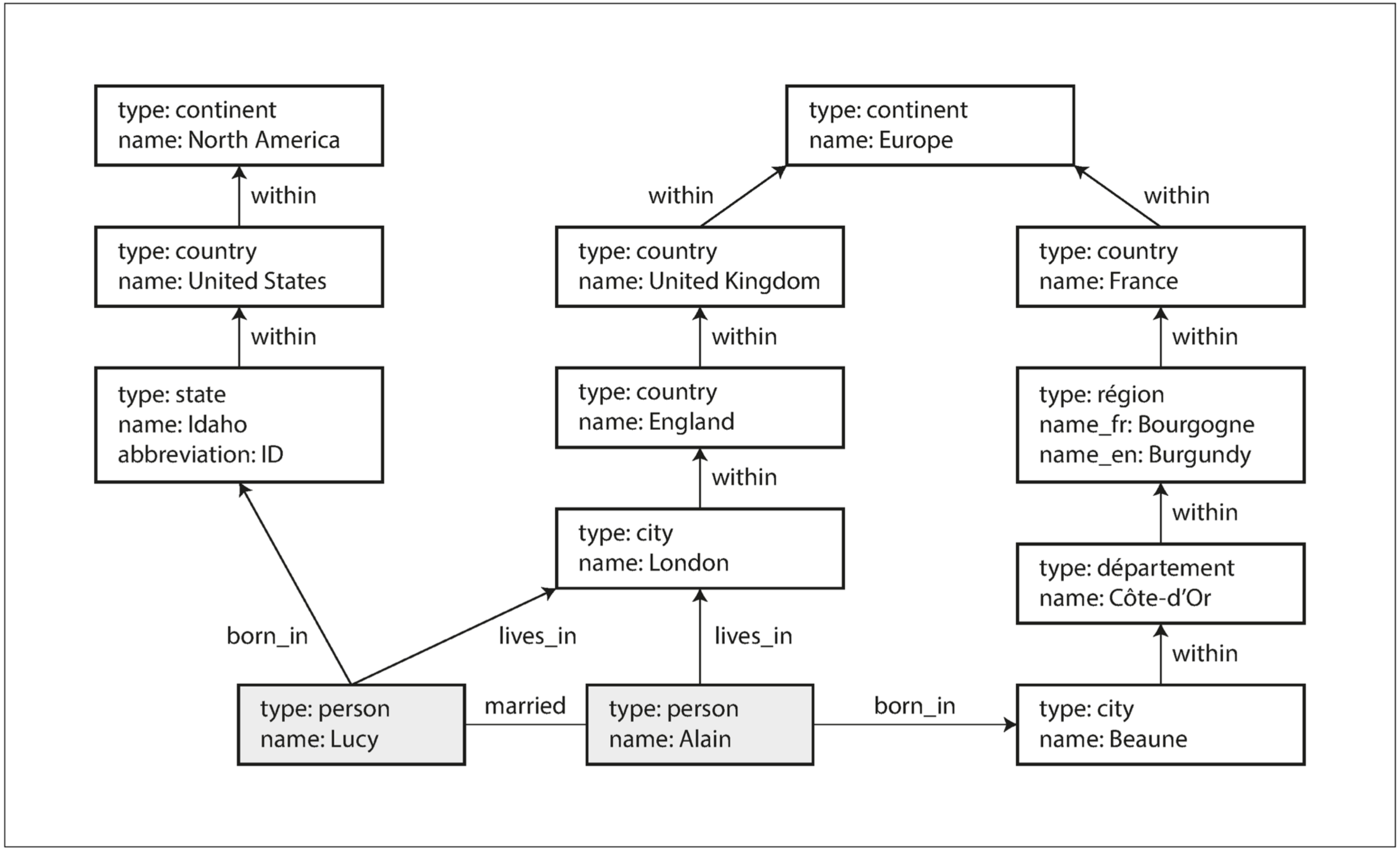
Imperative languages for graphs

- Gremlin

Environments for graph processing

- Pregel

Social network example



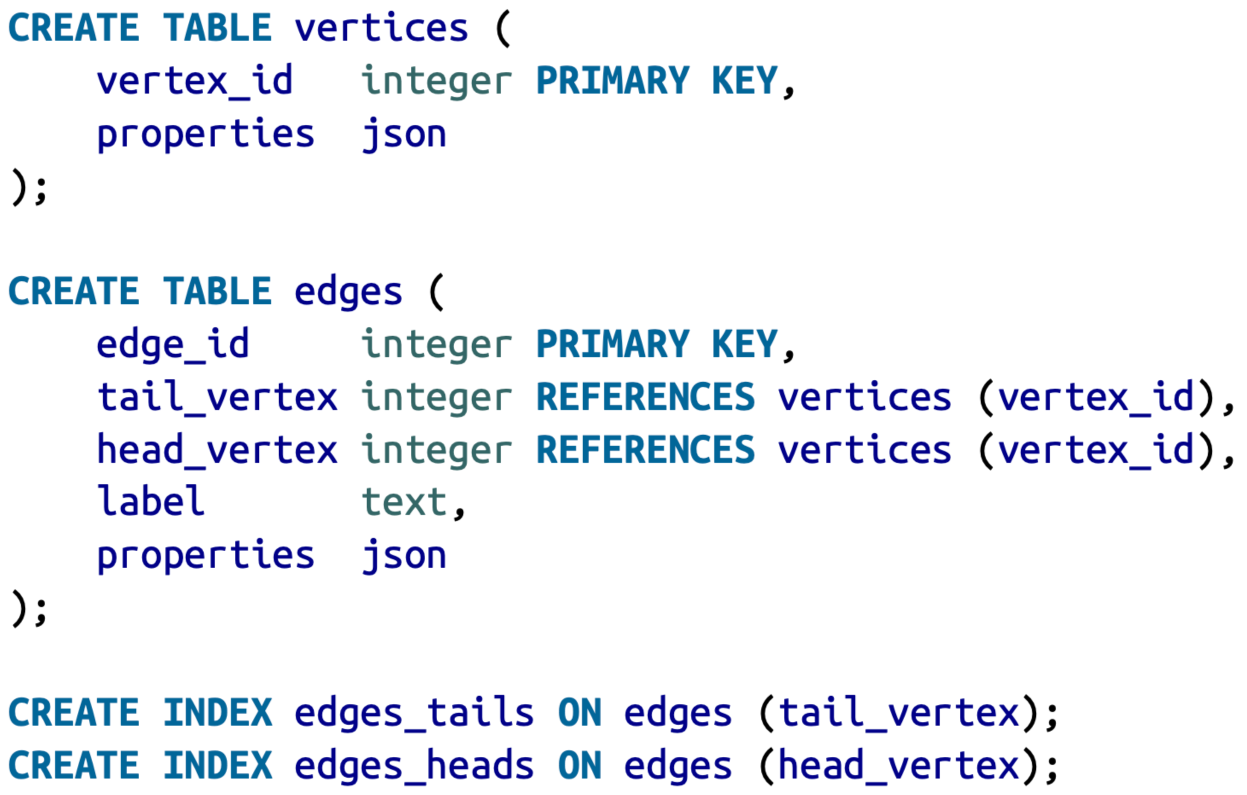
Property graph: tables

Node/vertex

- Node identifier
- Output links
- Input links
- Properties (key-value)

Link/edge

- Edge identifier
- initial node –> final node
- relationship description
- Properties (key-value)



```sql
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);
```

It’s easy to express complex relationships

- Each node can be connected with any other with a new link
- Graph **can be traversed** using links
- Use links to label **different kinds of relationships** in the same graph

- Example: we can adapt to different regional structures of countries
  - USA: counties and states
  - France: departements and regions
- Good for evolvability: as you add features to your application, a graph can easily be extended to accommodate changes in data structures.
  - Easy to add new functionality with links, nodes and labels

How to describe entities

- Describe a place on earth

```
(USA:Location {name: ‘United States’, type: ‘country’ } )
(California:Location {name ‘California’, type: ‘state’} )
```

- Describe a person

```
(Person:Lucy {name: ‘Lucy’} )
```

Describe entity relationships

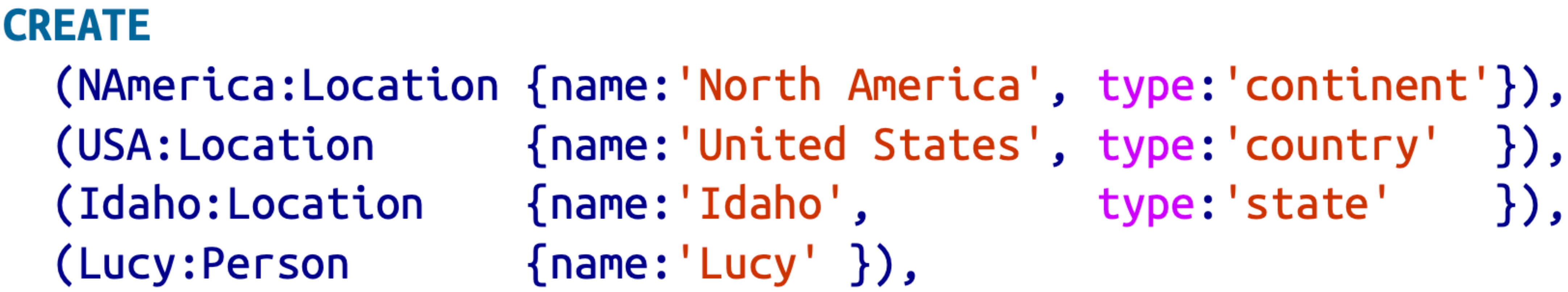
- California is a US state

```
(California) -[:WITHIN]-> (USA)
```

- Lucy was born in California

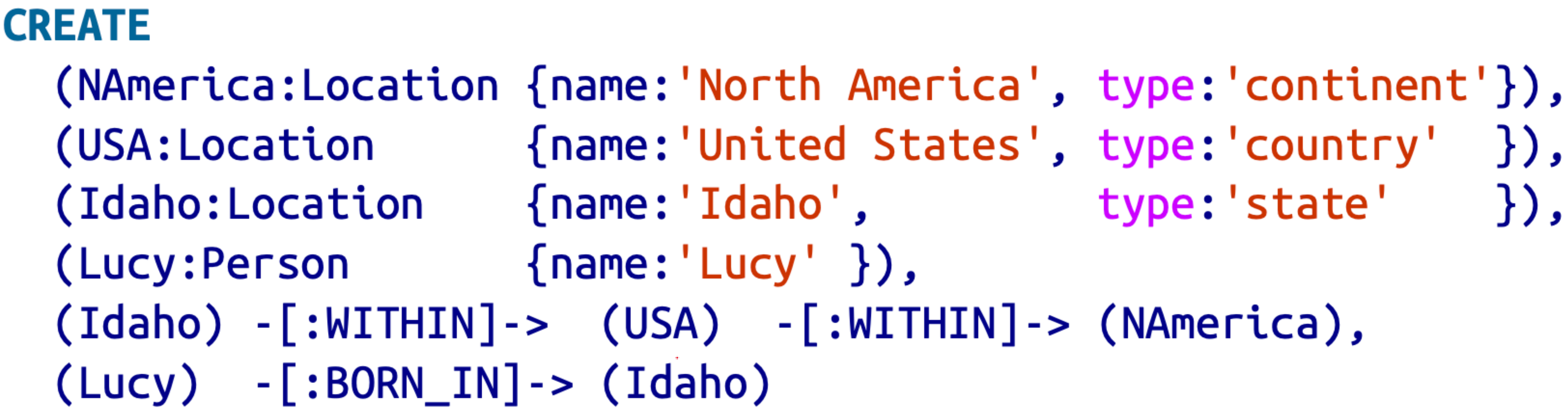
```
(Lucy) -:[BORN_IN]-> (California)
```

Cypher language for graphs



```
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' }),
```

Cypher language for graphs



```
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)
```

Cypher graf model queries

Names of people who went from US to Europe?

search for nodes with these links:

- **BORN_IN** link pointing to USA
- **LIVING_IN** link pointing to Europe

- Our objective is to extract a **specific property**:
  - **name** property

person BORN_IN WITHIN Location USA

Find a node (person) that meets condition:

1-person has a BORN_IN link that:

has WITHIN links that can be followed to

a node of Location type

with a name property equal to “USA”

person LIVES_IN WITHIN Location Europe

Finds a node (person) that follows two conditions:

1-person has a BORN_IN link with a number of WITHIN links that connect to a node of Location type with a name property equal to “United States”

2-the same person node has a LIVES_IN link with any number of WITHIN links that connect to a node of Location type with name = ‘Europe’

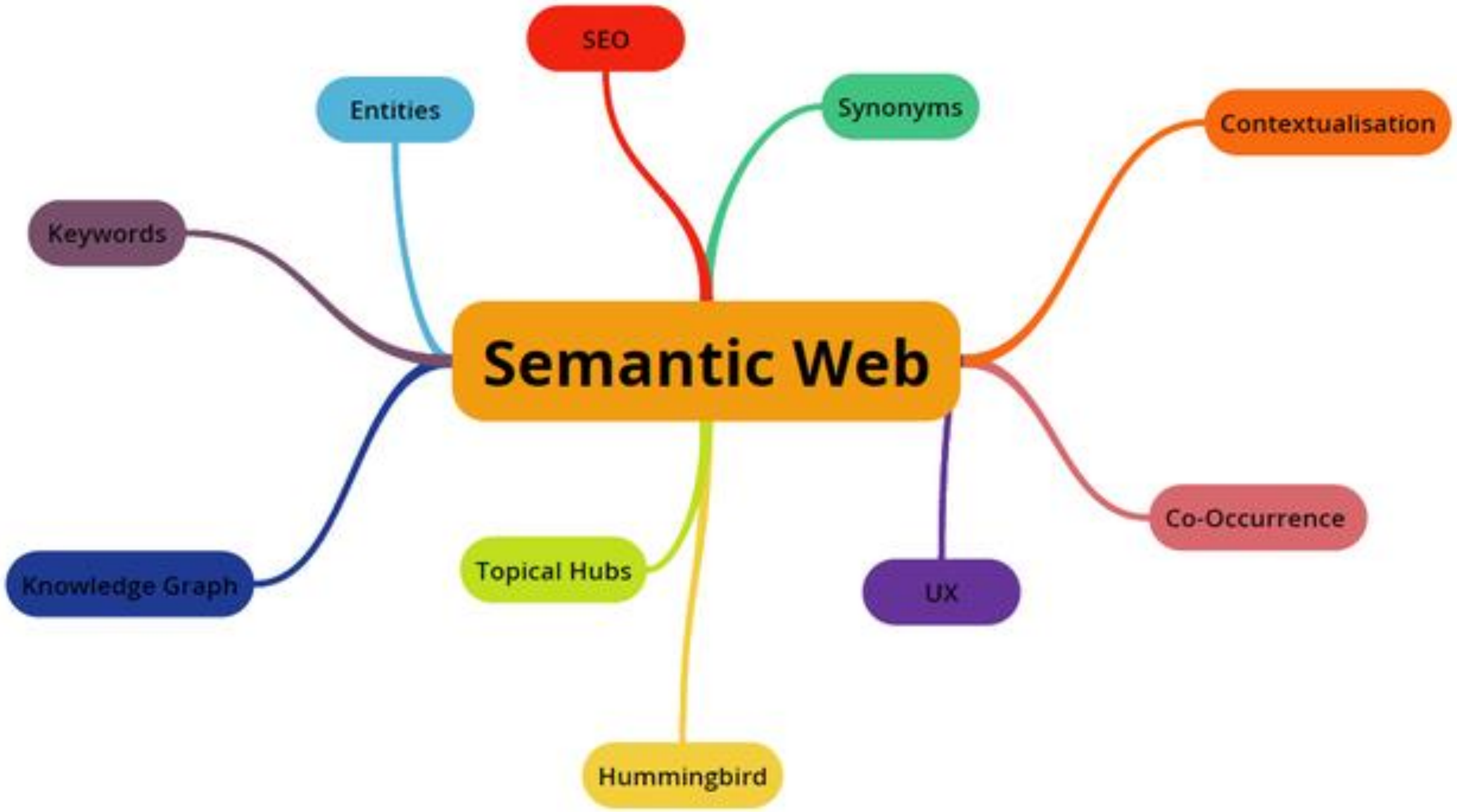
Cypher MATCH

1-person has a BORN_IN link with a number of WITHIN links that connect to a node of Location type with a name property equal to “United States”

2-the same person node has a LIVES_IN link with any number of WITHIN links that connect to a node of Location type with name = ‘Europe’

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

Web as a database



Semantic web

- Websites publish information for humans (HTML)
- Data web: why not publish information for machines?
- Web seen as a largely distributed DataBase

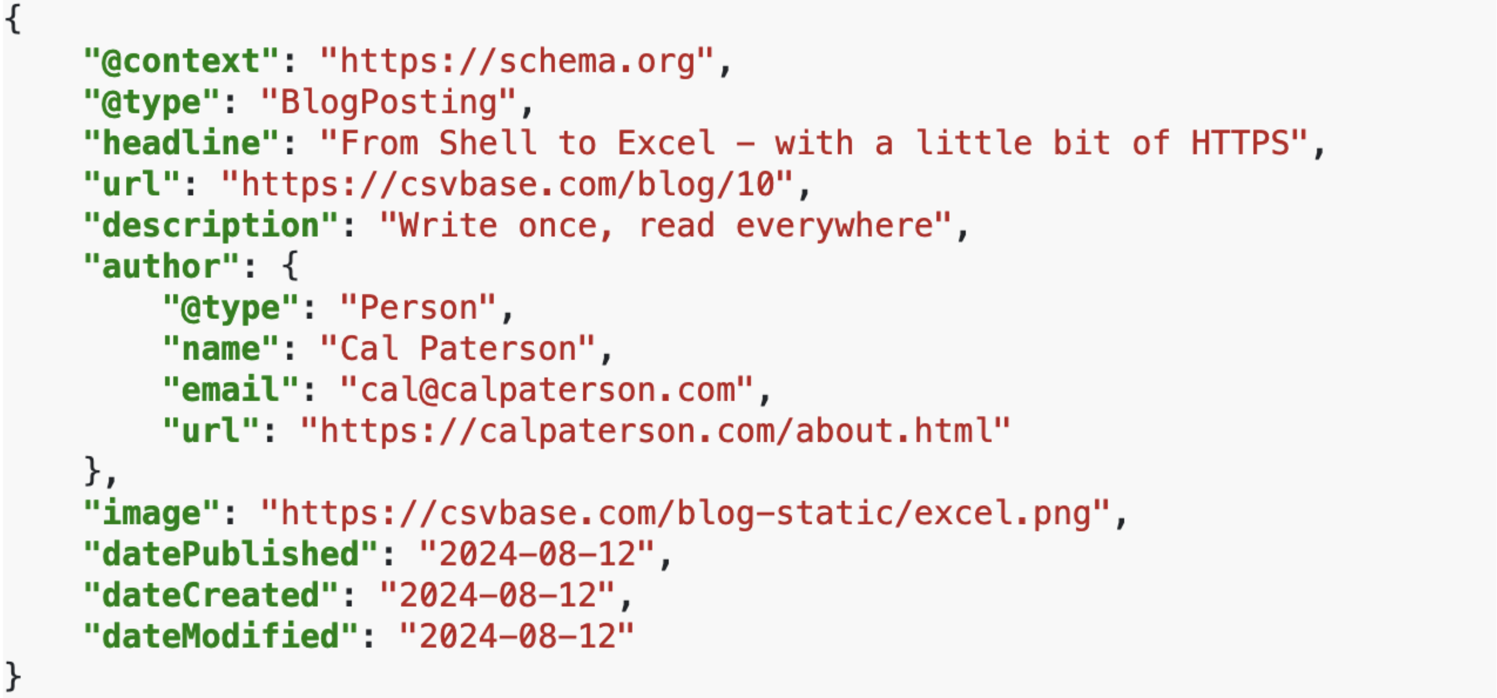
- Resource Description Framework (RDF): mechanism for publishing data in a consistent format

RDF example / XML syntax

```xml
<rdf:RDF xmlns="urn:example:"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

  <Location rdf:nodeID="idaho">
    <name>Idaho</name>
    <type>state</type>
    <within>
      <Location rdf:nodeID="usa">
        <name>United States</name>
        <type>country</type>
        <within>
          <Location rdf:nodeID="namerica">
            <name>North America</name>
            <type>continent</type>
          </Location>
        </within>
      </Location>
    </within>
  </Location>

  <Person rdf:nodeID="lucy">
    <name>Lucy</name>
    <bornIn rdf:nodeID="idaho"/>
  </Person>
</rdf:RDF>
```

Metadata with JSON-LD

- Machine-readable
- Context: how to interpret data
  - @id, @type, @language, @value, …
- <script type="application/ld+json"> element inside the <head> of an HTML page
- *Permissionless:* anyone can take the metadata and run with it.



```
{
    "@context": "https://schema.org",
    "@type": "BlogPosting",
    "headline": "From Shell to Excel - with a little bit of HTTPS",
    "url": "https://csvbase.com/blog/10",
    "description": "Write once, read everywhere",
    "author": {
        "@type": "Person",
        "name": "Cal Paterson",
        "email": "cal@calpaterson.com",
        "url": "https://calpaterson.com/about.html"
    },
    "image": "https://csvbase.com/blog-static/excel.png",
    "datePublished": "2024-08-12",
    "dateCreated": "2024-08-12",
    "dateModified": "2024-08-12"
}
```

What is JSON-LD? https://www.youtube.com/watch?v=vioCbTo3C-4

reference:

Chapter 2: Data Models and Query Languages

“**Designing Data Intensive Applications”**

MARTIN KLEPPMAN – O’REILLY – 2017