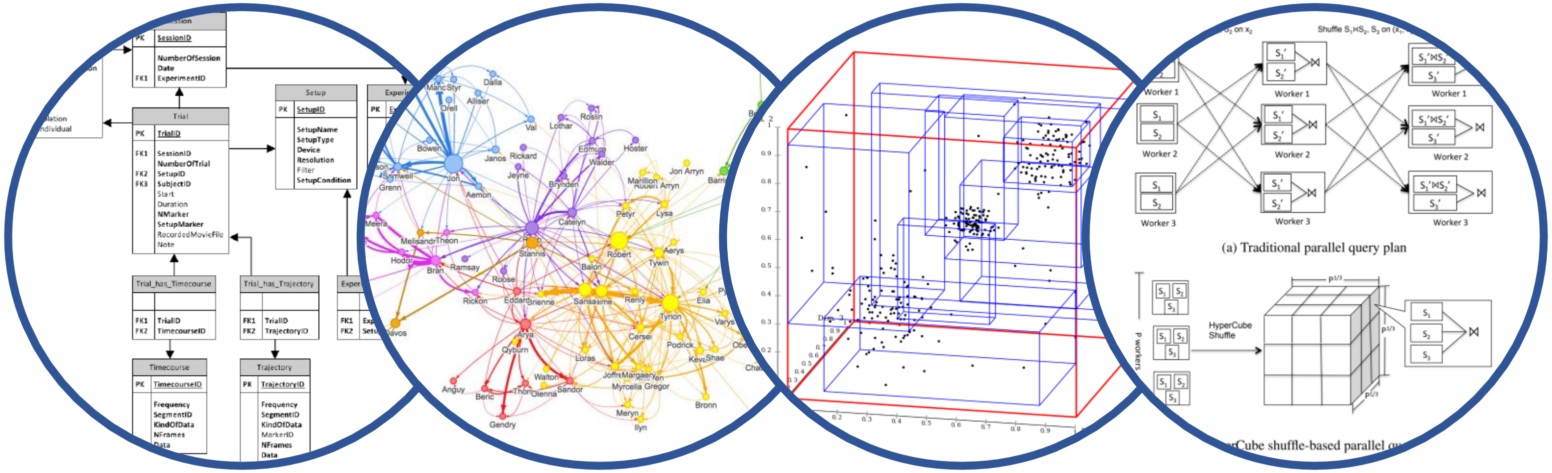


Introduction to Data Management

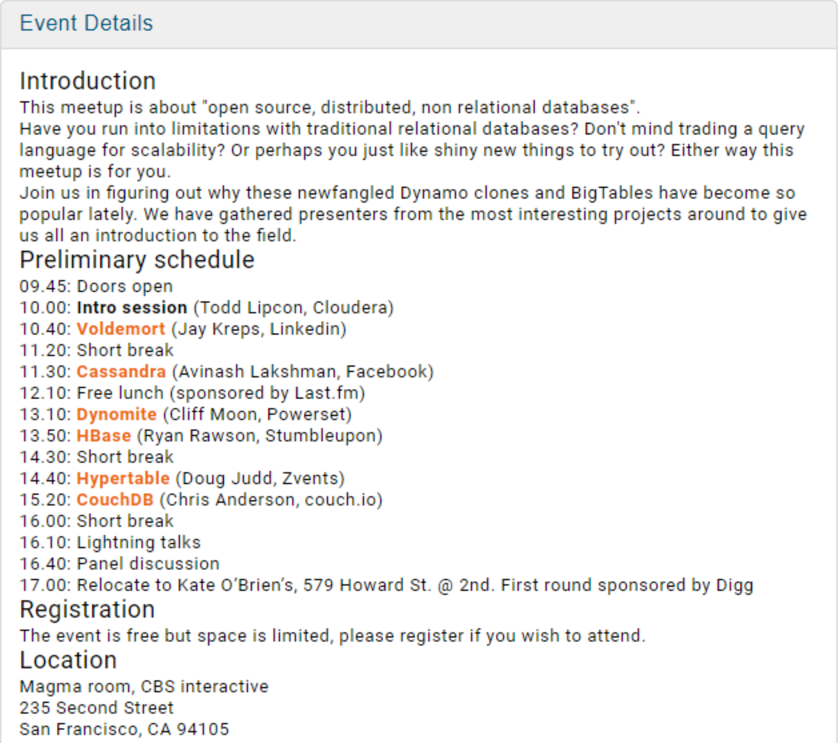
JSON, AsterixDB, and SQL++
Your First Non-Relational Data Model

Jonathan Leang

Paul G. Allen School of Computer Science and Engineering
University of Washington, Seattle

Recap: #NoSQL

A hashtag on Twitter for a meetup in San Francisco to discuss systems like Google BigTable, Amazon Dynamo, CouchDB, etc.



Event Details

Introduction

This meetup is about "open source, distributed, non relational databases".
Have you run into limitations with traditional relational databases? Don't mind trading a query language for scalability? Or perhaps you just like shiny new things to try out? Either way this meetup is for you.
Join us in figuring out why these newfangled Dynamo clones and BigTables have become so popular lately. We have gathered presenters from the most interesting projects around to give us all an introduction to the field.

Preliminary schedule

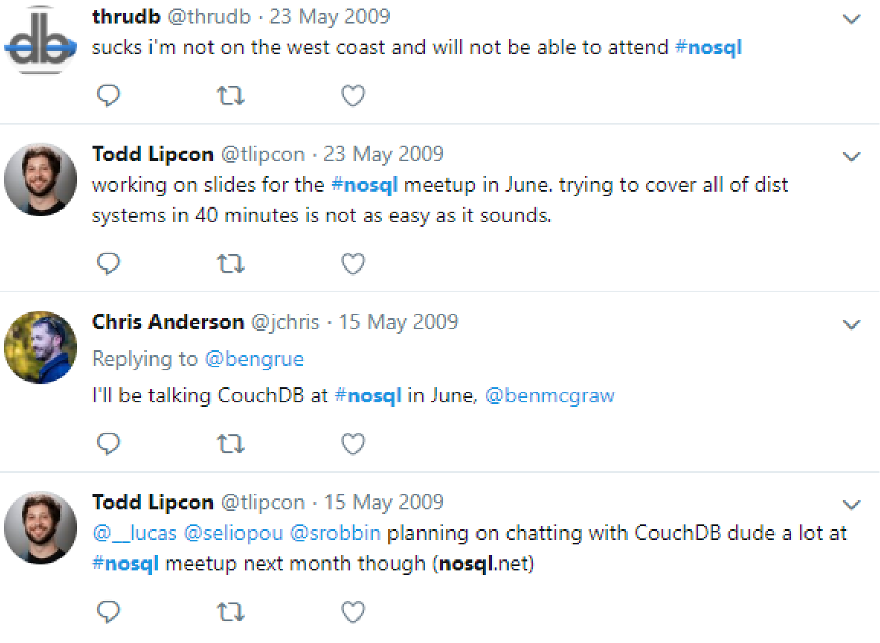
09.45: Doors open
10.00: **Intro session** (Todd Lipcon, Cloudera)
10.40: **Voldemort** (Jay Kreps, Linkedin)
11.20: Short break
11.30: **Cassandra** (Avinash Lakshman, Facebook)
12.10: Free lunch (sponsored by Last.fm)
13.10: **Dynomite** (Cliff Moon, Powerset)
13.50: **HBase** (Ryan Rawson, Stumbleupon)
14.30: Short break
14.40: **Hypertable** (Doug Judd, Zvents)
15.20: **CouchDB** (Chris Anderson, couch.io)
16.00: Short break
16.10: Lightning talks
16.40: Panel discussion
17.00: Relocate to Kate O'Brien's, 579 Howard St. @ 2nd. First round sponsored by Digg

Registration

The event is free but space is limited, please register if you wish to attend.

Location

Magma room, CBS interactive
235 Second Street
San Francisco, CA 94105



thrudb @thrudb · 23 May 2009
sucks i'm not on the west coast and will not be able to attend **#nosql**

Todd Lipcon @tlipcon · 23 May 2009
working on slides for the **#nosql** meetup in June. trying to cover all of dist systems in 40 minutes is not as easy as it sounds.

Chris Anderson @jchris · 15 May 2009
Replying to @bengrue
I'll be talking CouchDB at **#nosql** in June. @benmcgraw

Todd Lipcon @tlipcon · 15 May 2009
@__lucas @seliopou @srobbin planning on chatting with CouchDB dude a lot at **#nosql** meetup next month though (**nosql**.net)

Recap: The Modern World Wide Web

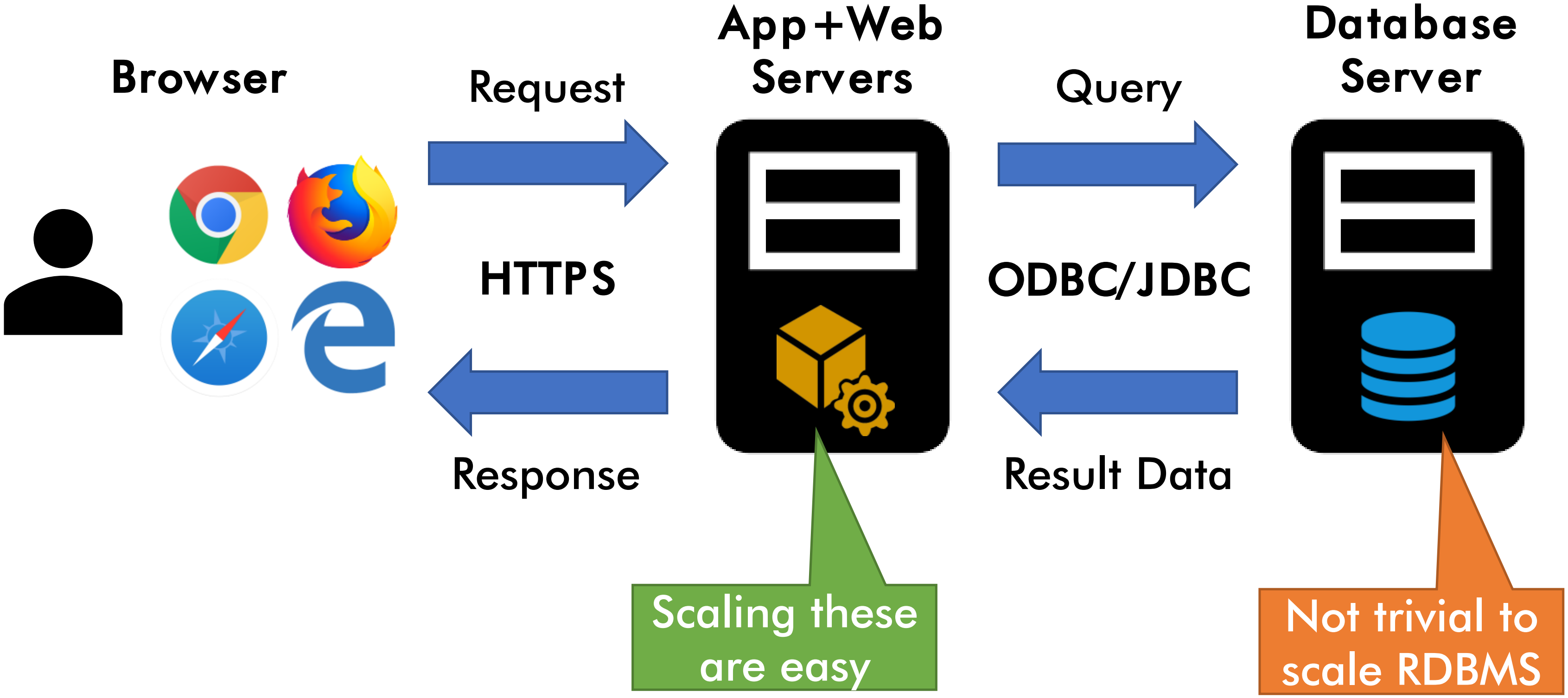
- What is Web 2.0?
  - Social Web
  - Everyone making content → **Everyone making data**
  - Facebook, Amazon, Instagram, …

- Web 2.0 problems are **specific**
  - Almost always OLTP-like workloads

- Web 2.0 problems are **big**
  - Data can’t fit into a single machine

Recap: Classic RDBMS for Web 2.0

- 3-Tier Web Apps (in a nutshell)
  - You (browsers) send requests to App+Web Servers
  - App+Web Servers send queries to a DB Server

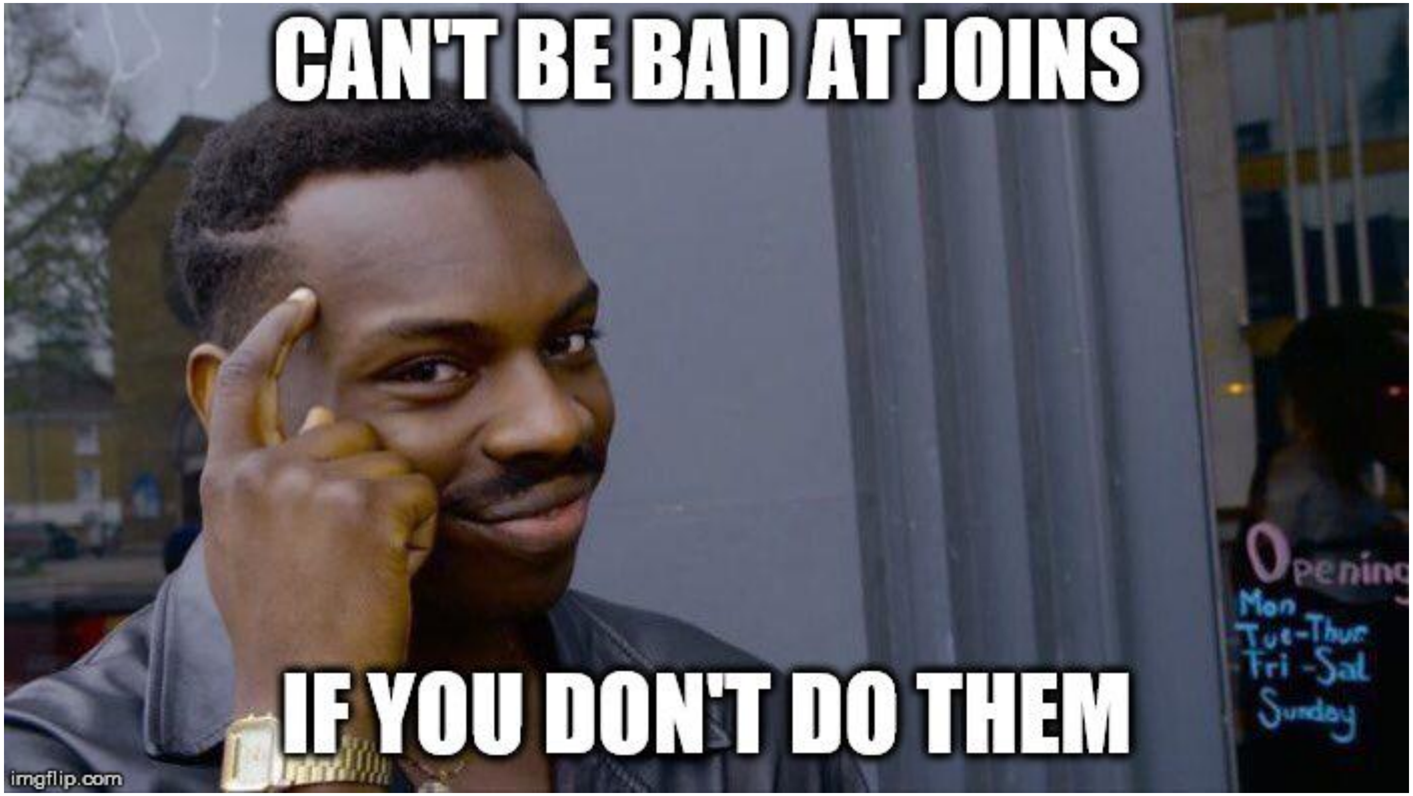


Recap: NoSQL on the Scale Up Problem

Recap: NoSQL on the Scale Up Problem

i give up

Recap: NoSQL on the Scale Up Problem



Recap: NoSQL on the Scale Up Problem

- KV Store
  - Hash Table (Key → Blob)
- Extensible Records
  - "2D" Hash Table (Row → Column → Blob)
- Document Store
  - Hash Table + Parsable Documents

Trade off well-defined data for speed

Recap: NoSQL on the Scale Up Problem

- KV Store
  - Hash Table (Key → Blob)
- Extensible Records
  - "2D" Hash Table (Row → Column → Blob)
- Document Store
  - Hash Table + Parsable Documents

Take Distributed Systems (CSE 452)

Trade off well-defined data for speed

Recap: NoSQL on the Scale Up Problem

- KV Store
  - Hash Table (Key → Blob)
- Extensible Records
  - "2D" Hash Table (Row → Column → Blob)
- Document Store
  - Hash Table + Parsable Documents

Good discussion for this class

Trade off well-defined data for speed

Recap: 3 Parts of a Data Model

The 3 parts of any data model

- Instance
  - The actual **data**
- Schema
  - A **description** of what data is being stored
- Query Language
  - How to retrieve and manipulate data

Today

Last time:

- Survey of NoSQL systems

Today

- AsterixDB as a case study of Document Store
  - Semi-structured data model in JSON
  - Introducing AsterixDB and SQL++

Today

Last time:

- Survey of NoSQL systems

Today

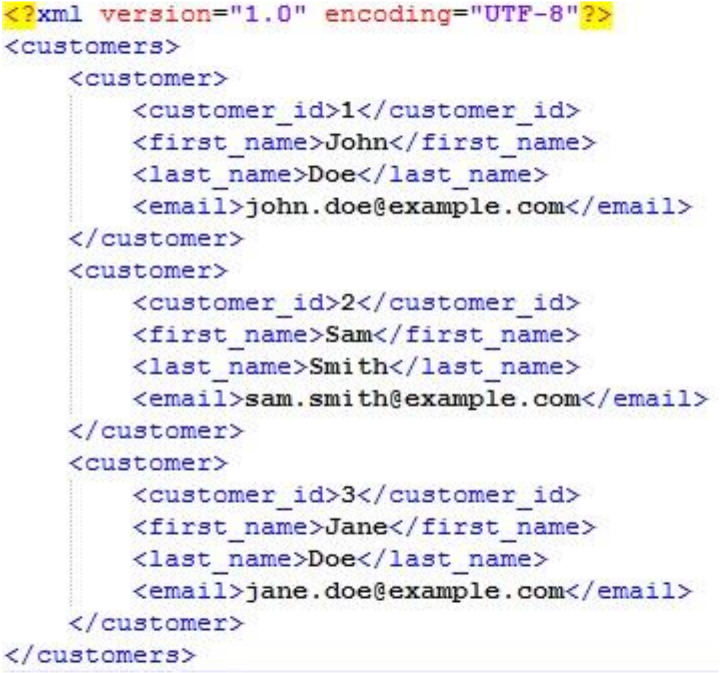
- AsterixDB as a case study of Document Store
  - **Semi-structured data model in JSON**
  - Introducing AsterixDB and SQL++

What is a "document" anyways?

- Loose terminology
- Any "parsable" file qualifies
  - Ex: MongoDB can handle CSV files

Semi-Structured Documents

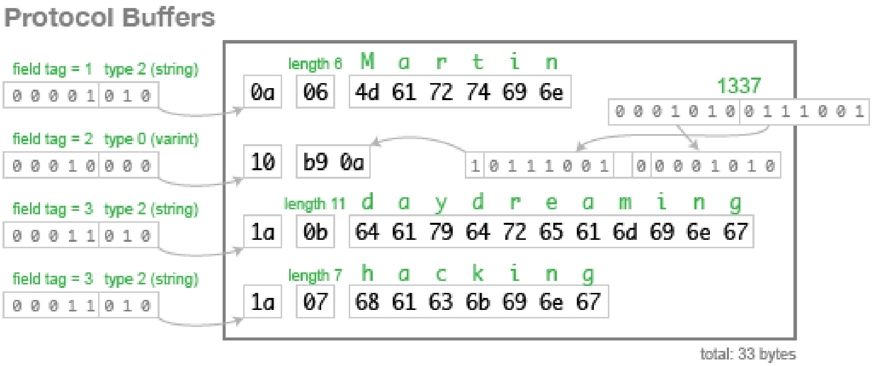
- Some notion of **tagging** to mark down semantics
- Examples:
  - XML
  - Protobuf
  - Email
  - JSON



```xml
<?xml version="1.0" encoding="UTF-8"?>
<customers>
    <customer>
        <customer_id>1</customer_id>
        <first_name>John</first_name>
        <last_name>Doe</last_name>
        <email>john.doe@example.com</email>
    </customer>
    <customer>
        <customer_id>2</customer_id>
        <first_name>Sam</first_name>
        <last_name>Smith</last_name>
        <email>sam.smith@example.com</email>
    </customer>
    <customer>
        <customer_id>3</customer_id>
        <first_name>Jane</first_name>
        <last_name>Doe</last_name>
        <email>jane.doe@example.com</email>
    </customer>
</customers>
```

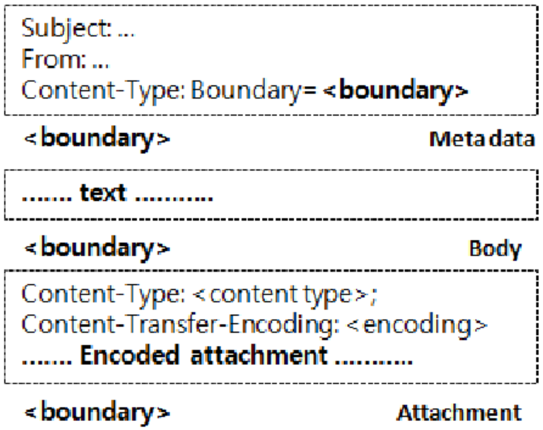
Semi-Structured Documents

- Some notion of **tagging** to mark down semantics
- Examples:
  - XML
  - **Protobuf**
  - Email
  - JSON



Semi-Structured Documents

- Some notion of **tagging** to mark down semantics
- Examples:
  - XML
  - Protobuf
  - **Email**
  - JSON



Subject: ...
From: ...
Content-Type: Boundary= <**boundary**>

<**boundary**> **Metadata**

.... **text** ....

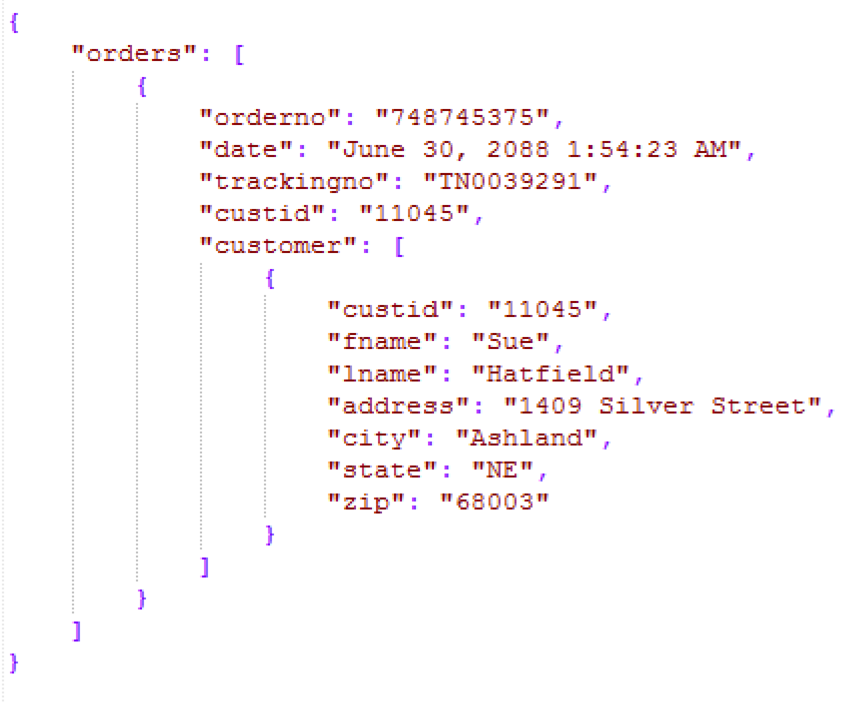
<**boundary**> **Body**

Content-Type: <content type>;
Content-Transfer-Encoding: <encoding>
.... **Encoded attachment** ....

<**boundary**> **Attachment**

Semi-Structured Documents

- Some notion of **tagging** to mark down semantics
- Examples:
  - XML
  - Protobuf
  - Email
  - JSON



```
{
    "orders": [
        {
            "orderno": "748745375",
            "date": "June 30, 2088 1:54:23 AM",
            "trackingno": "TN0039291",
            "custid": "11045",
            "customer": [
                {
                    "custid": "11045",
                    "fname": "Sue",
                    "lname": "Hatfield",
                    "address": "1409 Silver Street",
                    "city": "Ashland",
                    "state": "NE",
                    "zip": "68003"
                }
            ]
        }
    ]
}
```

Relational vs Semi-Structured Tradeoffs

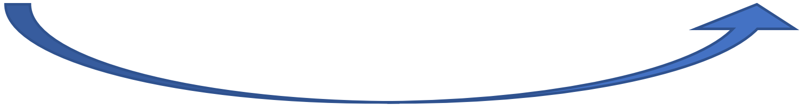
- Relational Model
  - Fixed schema
  - Flat data

- Semi-Structured
  - Self-described schema
  - Tree-structured data

Relational vs Semi-Structured Tradeoffs

- Relational Model
  - Fixed schema
  - Flat data

- Semi-Structured
  - Self-described schema
  - Tree-structured data

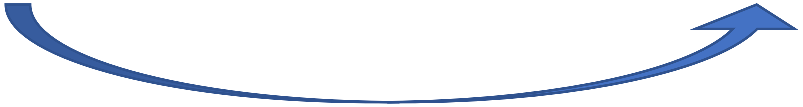


Less well-defined/More flexible

Relational vs Semi-Structured Tradeoffs

- **Relational Model**
  - Fixed schema
  - Flat data

- **Semi-Structured**
  - Self-described schema
  - Tree-structured data



Less well-defined/More flexible

- Basic retrieval process:
  1. Get table with all possible data
  2. Run through rows
  3. Return data

- Basic retrieval process:
  1. Get document with specific data
  2. Parse document tree
  3. Return data

Relational vs Semi-Structured Tradeoffs

- **Relational Model**
  - Fixed schema
  - Flat data

- **Semi-Structured**
  - Self-described schema
  - Tree-structured data

Less well-defined/More flexible

- Basic retrieval process:
  1. Get table with all possible data
  2. Run through rows
  3. Return data

- Basic retrieval process:
  1. Get document with specific data
  2. Parse document tree
  3. Return data

Inefficient encoding/Easy exchange of data

On A Practical Note

- No database paradigm is "better" than another
- One-size does **not** fit all (M. Stonebraker)
- Everything is getting mixed up anyways

On A Practical Note

- No database paradigm is "better" than another
- One-size does **not** fit all (M. Stonebraker)
- Everything is getting mixed up anyways

JSON Standard – Rules of the Game

- JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": "second",
         "price": 22.25
      }
   ]
}
```

JSON Standard – Rules of the Game

- JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
    "book":[
       {
          "id": "01",
          "language": "Java",
          "author": "H. Javeson",
          "year": 2015
       },
       {
          "author": "E. Sepp",
          "id": "07",
          "language": "C++",
          "edition": "second",
          "price": 22.25
       }
    ]
}
```

Types

Primitives include:

- String (in quotes)
- Numeric (unquoted number)
- Boolean (unquoted true/false)
- Null (literally just null)

JSON Standard – Rules of the Game

- **JavaScript Object Notation (JSON)**
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": "second",
         "price": 22.25
      }
   ]
}
```

Types

Objects are an *unordered* collection of name-value pairs:

- "name": <value>
- Values can be any type
- Enclosed by { }

JSON Standard – Rules of the Game

- JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
    "book":[
        {
            "id": "01",
            "language": "Java",
            "author": "H. Javeson",
            "year": 2015
        },
        {
            "author": "E. Sepp",
            "id": "07",
            "language": "C++",
            "edition": "second",
            "price": 22.25
        }
    ]
}
```

Types

Objects are an *unordered* collection of name-value pairs:

- "name": <value>
- Values can be any type
- Enclosed by { }

JSON Standard – Rules of the Game

- JavaScript Object Notation (JSON)
  - "Lightweight text-based open standard designed for **human-readable** data interchange"

```
{
   "book":[
      {
         "id": "01",
Index 0  "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
Index 1  "language": "C++",
         "edition": "second",
         "price": 22.25
      }
   ]
}
```

Types

Arrays are an *ordered* list of values:

- Order is preserved in interpretation
- May contain any mix of types
- Enclosed by [ ]

JSON Standard – Rules of the Game

- **JSON Standard too expressive**
  - Implementations **restrict syntax**
  - Ex: Duplicate fields

```
{
   "id": "01",
   "language": "Java",
   "author": "H. Javeson",
   "author": "D. Suciu",
   "author": "A. Cheung",
   "year": 2015
}
```

JSON Standard – Rules of the Game

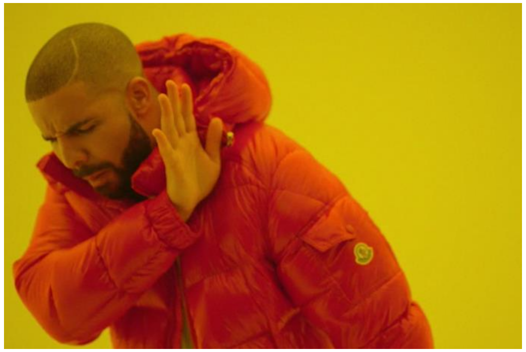
- **JSON Standard too expressive**
  - Implementations **restrict syntax**
  - Ex: Duplicate fields

```
{
    "id": "01",
    "language": "Java",
    "author": "H. Javeson",
    "author": "D. Suciu",
    "author": "A. Cheung",
    "year": 2015
}
```

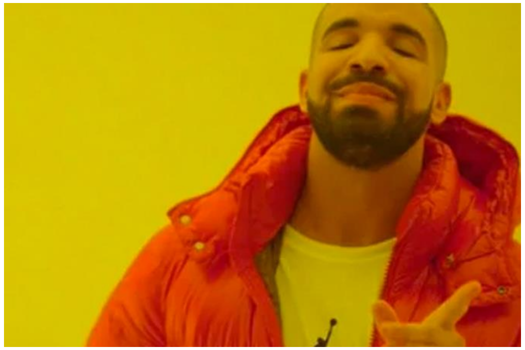
```
{
    "id": "01",
    "language": "Java",
    "author": ["H. Javeson",
               "D. Suciu",
               "A. Cheung"]
    "year": 2015
}
```

JSON Standard – Rules of the Game

- **JSON Standard too expressive**
  - Implementations **restrict syntax**
  - Ex: Duplicate fields



```
{
   "id": "01",
   "language": "Java",
   "author": "H. Javeson",
   "author": "D. Suciu",
   "author": "A. Cheung",
   "year": 2015
}
```



```
{
   "id": "01",
   "language": "Java",
   "author": ["H. Javeson",
              "D. Suciu",
              "A. Cheung"]
   "year": 2015
}
```

Thinking About Semi-Structured Data

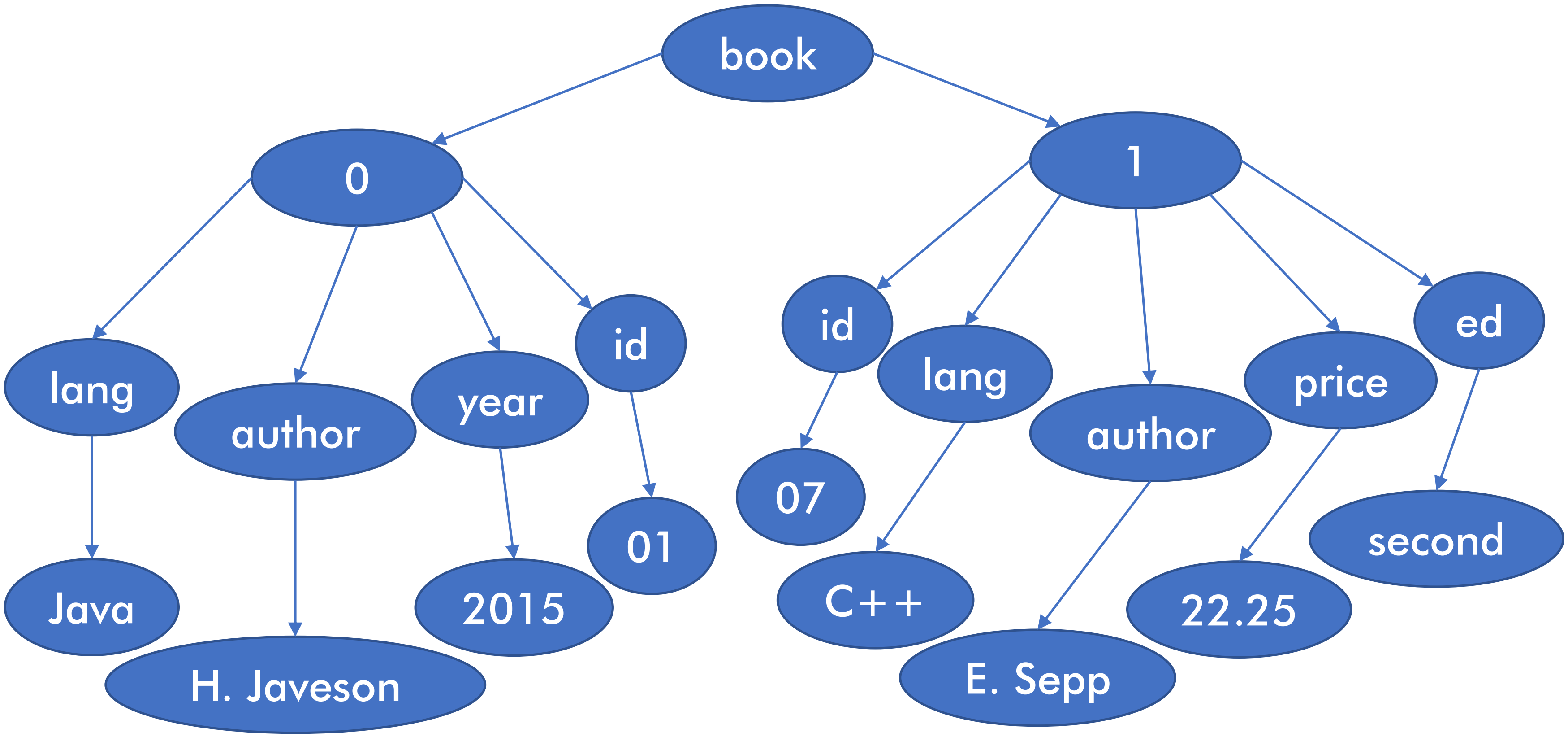
What does semi-structured data structure encode?

```
{
   "book":[
      {
         "id": "01",
         "language": "Java",
         "author": "H. Javeson",
         "year": 2015
      },
      {
         "author": "E. Sepp",
         "id": "07",
         "language": "C++",
         "edition": "second",
         "price": 22.25
      }
   ]
}
```

Thinking About Semi-Structured Data

What does semi-structured data structure encode?

Tree semantics!



From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

What is a table in semi-structured land?

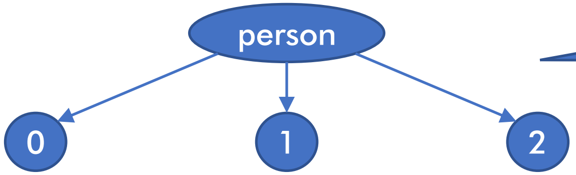
person

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

What is a table in semi-structured land?



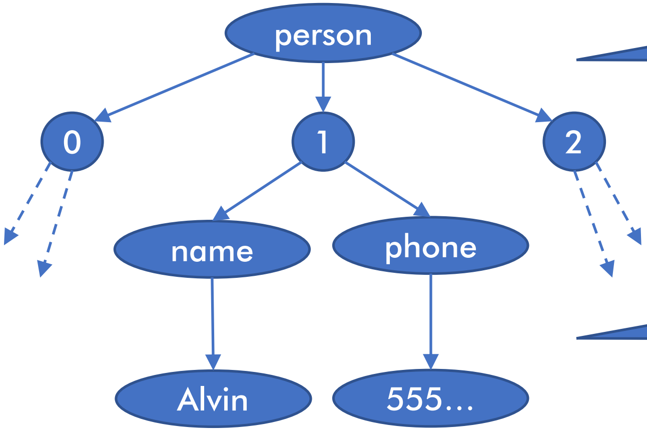
Tables are just an array of elements (rows)

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

What is a table in semi-structured land?



Tables are just an array of elements (rows)

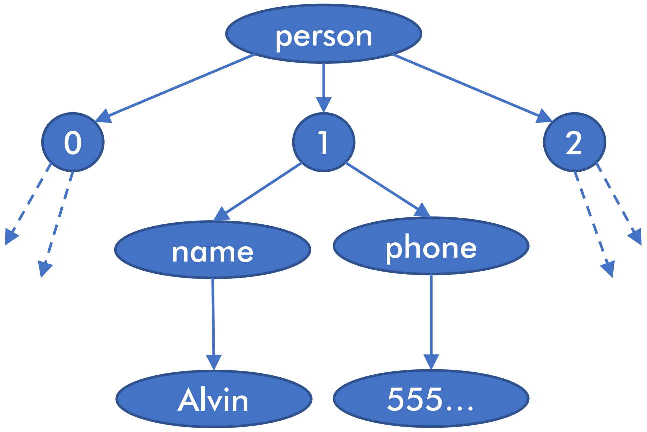
Rows are just simple objects

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

```
{
    "person":[
        {
            "name": "Dan",
            "phone": "555-123-4567"
        },
        {
            "name": "Alvin",
            "phone": "555-234-5678"
        },
        {
            "name": "Magda",
            "phone": "555-345-6789"
        },
    ]
}
```



From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      },
   ]
}
```

How can NULL
be represented?

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	NULL

How can NULL
be represented?

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      },
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	NULL

How can NULL
be represented?

```
{
    "person":[
        {
            "name": "Dan",
            "phone": "555-123-4567"
        },
        {
            "name": "Alvin",
            "phone": "555-234-5678"
        },
        {
            "name": "Magda",
            "phone": null
        },
    ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	NULL

How can NULL be represented?

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda"
      },
   ]
}
```

OK for field to be missing!

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Are there things that the Relational Model can't represent?

```
{
    "person":[
        {
            "name": "Dan",
            "phone": "555-123-4567"
        },
        {
            "name": "Alvin",
            "phone": "555-234-5678"
        },
        {
            "name": "Magda",
            "phone": "555-345-6789"
        },
    ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

```
{
    "person":[
        {
            "name": "Dan",
            "phone": "555-123-4567"
        },
        {
            "name": "Alvin",
            "phone": "555-234-5678"
        },
        {
            "name": "Magda",
            "phone": "555-345-6789"
        },
    ]
}
```

Are there things that the Relational Model can't represent?

Non-flat data!

- Array data
- Multi-part data

From Relational to Semi-Structured

Person

Name	Phone
Dan	???
Alvin	555-234-5678
Magda	555-345-6789

Are there things that the Relational Model can't represent?

Non-flat data!

- **Array data**
- Multi-part data

```
{
   "person":[
      {
         "name": "Dan",
         "phone": [
            "555-123-4567",
            "555-987-6543"
         ]
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      },
   ]
}
```

From Relational to Semi-Structured

Person

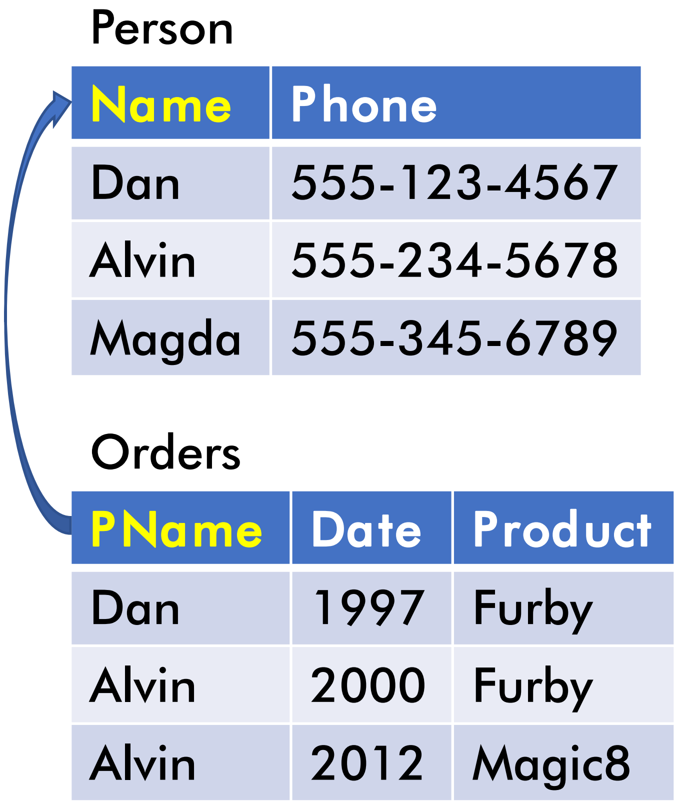
Name	Phone
???	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Are there things that the Relational Model can’t represent?

Non-flat data!
- Array data
- **Multi-part data**

```
{
   "person":[
      {
         "name": {
            "fname": "Dan",
            "lname": "Suciu"
         }
         "phone": "555-123-4567"
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678"
      },
      {
         "name": "Magda",
         "phone": "555-345-6789"
      },
   ]
}
```

From Relational to Semi-Structured



Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

How do we represent foreign keys?

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567",
         "orders": [
            {
               "date": 1997,
               "product": "Furby",
            }
         ]
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678",
         "orders": [
            {
               "date": 2000,
               "product": "Furby",
            },
            {
               "date": 2012,
               "product": "Magic8",
            }
         ]
      },
      {
         "name": "Magda",
         "phone": "555-345-6789",
         "orders": []
      },
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Precomputed equijoin!

```
{
   "person":[
      {
         "name": "Dan",
         "phone": "555-123-4567",
         "orders": [
            {
               "date": 1997,
               "product": "Furby",
            }
         ]
      },
      {
         "name": "Alvin",
         "phone": "555-234-5678",
         "orders": [
            {
               "date": 2000,
               "product": "Furby",
            },
            {
               "date": 2012,
               "product": "Magic8",
            }
         ]
      },
      {
         "name": "Magda",
         "phone": "555-345-6789",
         "orders": []
      },
   ]
}
```

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this many-to-many relationship easily convertible to JSON?

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this many-to-many relationship easily convertible to JSON?

Nest the data?
Person → Orders → Product

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this many-to-many relationship easily convertible to JSON?

Nest the data?
Person → Orders → Product

We might miss some products!

Product data will be duplicated!

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this many-to-many relationship easily convertible to JSON?

Nest the data?
Product → Orders → Person

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this many-to-many relationship easily convertible to JSON?

Nest the data?
Product → Orders → Person

We might miss some people!

People data will be duplicated!

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this many-to-many relationship easily convertible to JSON?

Convert each table to a separate object/document?

From Relational to Semi-Structured

Person

Name	Phone
Dan	555-123-4567
Alvin	555-234-5678
Magda	555-345-6789

Orders

PName	Date	Product
Dan	1997	Furby
Alvin	2000	Furby
Alvin	2012	Magic8

Product

ProdName	Price
Furby	9.99
Magic8	15.99
Tomagachi	18.99

Is this many-to-many relationship easily convertible to JSON?

Convert each table to a separate object/document?

We wanted to avoid joining in the first place!

From Relational to Semi-Structured

Takeaways:

- Semi-structured data can do cool stuff
  - Collection/multi-part data
  - Precompute joins
- Semi-structured data has some limits
  - Relies on relational-like patterns in common situations
- In general semi-structured data is parsed
  - Data model flexibility
  - Potentially lots of redundancy

Today

- AsterixDB as a case study of Document Store
  - Semi-structured data model in JSON
  - **Introducing AsterixDB and SQL++**

The 5 W's of AsterixDB

- Who
  - M. J. Carey & co.
- What
  - "A Scalable, Open Source BDMS" (it is now also an Apache project)
- Where
  - UC Irvine, Cloudera Inc, Google, IBM, …
- When
  - 2014
- Why
  - To develop a next-gen system for managing semi-structured data

The 5 W's of SQL++

- Who
  - K. W. Ong & Y. Papakonstantinou
- What
  - A query language that is applicable to JSON native stores and SQL databases
- Where
  - UC San Diego
- When
  - 2015
- Why
  - Stand in for other semi-structured query languages that lack formal semantics.

Why We are Choosing SQL++

- Strong formal semantics
  - Original paper: https://arxiv.org/pdf/1405.3631.pdf
  - Nested relational algebra: https://dl.acm.org/citation.cfm?id=588133
- Systems adopting or converging to SQL++
  - Apache AsterixDB
  - CouchBase (N1QL)
  - Apache Drill
  - Snowflake

Asterix Data Model (ADM)

- Nearly Identical to JSON Standard
  - All JSON primitives
  - JSON objects and arrays
- Some additions
  - New primitive: **universally unique identifier (uuid)**
    - Ex: 123e4567-e89b-12d3-a456-426655440000
  - New derived type: **multiset**
    - Like an array but **unordered** and encapsulated by {{ }}
  - Missing (field not in object) is a thing
- Queried data must be a multiset or array

Introducing the New and Improved SQL++

SQL++ Mini Demo

Demo Time!

SQL++ Mini Demo

General Installation (Details in HW5 spec)

Download from:
https://asterixdb.apache.org/download.html

Start local cluster from:
<asterix root>/opt/local/bin/start-sample-cluster

Use web browser for interaction, default:
127.0.0.19001

Don’t forget to stop cluster when you’re done:
<asterix root>/opt/local/bin/stop-sample-cluster

SQL++ Mini Demo

General Usage:

Everything is running locally so make sure your computer doesn’t die (advise against SELECT *)

Don’t use attu, previous quarters people accidentally used other people’s instance

Learn something! I dare say that SQL++ is a model for many future query languages.

SQL++ Hello World

```
SELECT x.phone
  FROM [
          {"name": "Dan", "phone": [300, 150]},
          {"name": "Alvin", "phone": 420}
       ] AS x;

-- output, same for-loop semantics like in SQL
/*
{ "phone": [300, 150] }
{ "phone": 420 }
*/
```

SQL++ Hello World

```
SELECT x.phone
  FROM {{
          {"name": "Dan", "phone": [300, 150]},
          {"name": "Alvin", "phone": 420}
       }} AS x;

-- same output as array data
```

SQL++ Hello World

```
-- error
SELECT x.phone
  FROM {"name": "Dan", "phone": [300, 150]} AS x;

-- output
/*
Type mismatch: function scan-collection expects its
1st input parameter to be type multiset or array,
but the actual input type is object
[TypeMismatchException]
*/
```

SQL++ Hello World

```
SELECT x.phone
  FROM [
          {"name": "Dan", "phone": [300, 150]},
          {"name": "Alvin", "phone": null}
       ] AS x;

-- output, null works like in SQL
/*
{ "phone": [300, 150] }
{ "phone": null }
*/
```

SQL++ Hello World

```
SELECT x.phone
  FROM [
          {"name": "Dan", "phone": [300, 150]},
          {"name": "Alvin"}
       ] AS x;

-- output, missing data is simply passed over (beware of typos!)
/*
{ "phone": [300, 150] }
{ }
*/
```

SQL++ Hello World

```
SELECT x.fone -- intentional typo
  FROM [
          {"name": "Dan", "phone": [300, 150]},
          {"name": "Alvin", "phone": 420}
       ] AS x;

-- output, beware of typos!
/*
{ }
{ }
*/
```

SQL++ Hello World

```
SELECT x.fone -- intentional typo
  FROM [
          {"name": "Dan", "phone": [300, 150]},
          {"name": "Alvin", "phone": 420}
       ] AS x;

-- output, beware of typos!
/*
{ }
{ }
*/
```

SQL++ Hello World

```
  FROM [
          {"name": "Dan", "phone": [300, 150]},
          {"name": "Alvin", "phone": 420}
       ] AS x
 WHERE is_array(x.phone) OR x.phone > 100
 GROUP BY x.name, x.phone
HAVING x.name = "Dan" OR x.name = "Alvin"
SELECT x.phone
 ORDER BY x.name DESC;

-- output, finally the keyword order matches FWGHOS!
/*
{ "phone": [300, 150] }
{ "phone": 420 }
*/
```

Next Time

- Patterns in querying semi-structured data

- SQL++ behind the mask