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

February 3, 2019
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: Dystomite (Cliff Moon, Powerset)
13.50: HBase (Ryan Rawson, Stumbleupon)
14.30: Short break
14.40: Hypertable (Douq 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

February 3, 2019
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

- Scaling these are easy
- Not trivial to scale RDBMS
Recap: NoSQL on the Scale Up Problem

Modern problems require modern solutions
Recap: NoSQL on the Scale Up Problem

i give up
Recap: NoSQL on the Scale Up Problem

CAN'T BE BAD AT JOINS

IF YOU DON'T DO THEM
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

Trade off well-defined data for speed

Take Distributed Systems (CSE 452)
Recap: NoSQL on the Scale Up Problem

- **KV Store**
  - Hash Table (Key $\rightarrow$ Blob)

- **Extensible Records**
  - "2D" Hash Table (Row $\rightarrow$ Column $\rightarrow$ Blob)

- **Document Store**
  - Hash Table + Parsable Documents

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
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>Jan</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
Semi-Structured Documents

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

```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

**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

Less well-defined/More flexible

Inefficient encoding/Easy exchange of data
- No database paradigm is "better" than another
- One-size does not fit all (M. Stonebraker)
- Everything is getting mixed up anyways
- No database paradigm is "better" than another
- One-size does **not** fit all (M. Stonebraker)
- Everything is getting mixed up anyways
### JavaScript Object Notation (JSON)

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

```json
{
  "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 { }
JavaScript Object Notation (JSON)

- "Lightweight text-based open standard designed for human-readable data interchange"

Objects are an unordered collection of name-value pairs:
- "name": <value>
- Values can be any type
- Enclosed by { }
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
    }
  ]
}
```

Arrays are an ordered list of values:
- Order is preserved in interpretation
- May contain any mix of types
- Enclosed by [ ]
• JSON Standard too expressive
  • Implementations restrict syntax
  • Ex: Duplicate fields

```json
{
  "id": "01",
  "language": "Java",
  "author": "H. Javesson",
  "author": "D. Suciu",
  "author": "A. Cheung",
  "year": 2015
}
```
JSON Standard – Rules of the Game

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

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

```json
{  
  "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

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

```json
{
  "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?

```json
{
    "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!

- Book 0
  - Author: H. Javeson
    - Language: Java
    - Year: 2015
- Book 1
  - Author: E. Sepp
    - Language: C++
    - Price: 22.25
  - Edition: 07
  - Price: 22.25
What is a table in semi-structured land?

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

**Person**
From Relational to Semi-Structured

What is a table in semi-structured land?

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

Tables are just an array of elements (rows)
What is a table in semi-structured land?

Tables are just an array of elements (rows)

Rows are just simple objects

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>
From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

```
{  
  "person": [
    {
      "name": "Dan",
      "phone": "555-123-4567"
    },
    {
      "name": "Alvin",
      "phone": "555-234-5678"
    },
    {
      "name": "Magda",
      "phone": "555-345-6789"
    }
  ]
}
```
How can NULL be represented?

```json
{  
  "person" :  
  [  
    {  
      "name" : "Dan",  
      "phone" : "555-123-4567"  
    },  
    {  
      "name" : "Alvin",  
      "phone" : "555-234-5678"  
    },  
    {  
      "name" : "Magda",  
      "phone" : "555-345-6789"  
    }  
  ]
}
```
From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>NULL</td>
</tr>
</tbody>
</table>

How can NULL be represented?

```json
{  
  "person": [  
    {  
      "name": "Dan",  
      "phone": "555-123-4567"  
    },  
    {  
      "name": "Alvin",  
      "phone": "555-234-5678"  
    },  
    {  
      "name": "Magda",  
      "phone": "555-345-6789"  
    }  
  ]
}
```
How can NULL be represented?

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>NULL</td>
</tr>
</tbody>
</table>

```json

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>NULL</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

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

```json
{
    "person": [
        {
            "name": "Dan",
            "phone": "555-123-4567"
        },
        {
            "name": "Alvin",
            "phone": "555-234-5678"
        },
        {
            "name": "Magda",
            "phone": "555-345-6789"
        }
    ]
}
```
From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

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

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

```json
{
    "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

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>???</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

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

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

```json
Person
{
  "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

<table>
<thead>
<tr>
<th>Person</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orders</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PName</td>
<td>Date</td>
<td>Product</td>
</tr>
<tr>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
</tr>
</tbody>
</table>

How do we represent foreign keys?
From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

Orders

<table>
<thead>
<tr>
<th>PName</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
</tr>
</tbody>
</table>

```json
{
"person": ["Dan", "Alvin", "Magda"],
"orders": ["Furby", "Furby", "Magic8"],
"date": 1997, 2000, 2012
}
```
From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PName</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
</tr>
</tbody>
</table>

Precomputed equijoin!

```json
{
  "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": []
    }
  ]
}
```
Is this many-to-many relationship easily convertible to JSON?

<table>
<thead>
<tr>
<th>Person</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orders</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PName</td>
<td>Date</td>
<td>Product</td>
</tr>
<tr>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ProdName</td>
<td>Price</td>
</tr>
<tr>
<td>Furby</td>
<td>9.99</td>
</tr>
<tr>
<td>Magic8</td>
<td>15.99</td>
</tr>
<tr>
<td>Tomagachi</td>
<td>18.99</td>
</tr>
</tbody>
</table>
From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Person</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orders</th>
<th>PName</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>ProdName</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furby</td>
<td>Furby</td>
<td>9.99</td>
</tr>
<tr>
<td>Magic8</td>
<td>Magic8</td>
<td>15.99</td>
</tr>
<tr>
<td>Tomagachi</td>
<td>Tomagachi</td>
<td>18.99</td>
</tr>
</tbody>
</table>

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

Nest the data?
Person → Orders → Product
From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Person</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

Orders

<table>
<thead>
<tr>
<th>Person</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
</tr>
</tbody>
</table>

Product

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furby</td>
<td>9.99</td>
</tr>
<tr>
<td>Magic8</td>
<td>15.99</td>
</tr>
<tr>
<td>Tomagachi</td>
<td>18.99</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

#### Orders

<table>
<thead>
<tr>
<th>PName</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
</tr>
</tbody>
</table>

#### Product

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furby</td>
<td>9.99</td>
</tr>
<tr>
<td>Magic8</td>
<td>15.99</td>
</tr>
<tr>
<td>Tomagachi</td>
<td>18.99</td>
</tr>
</tbody>
</table>

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

Nest the data?
Product → Orders → Person
From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Person</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
<td></td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
<td></td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orders</th>
<th>PName</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
<td></td>
</tr>
<tr>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
<td></td>
</tr>
<tr>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>ProdName</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Furby</td>
<td>9.99</td>
<td></td>
</tr>
<tr>
<td>Magic8</td>
<td>15.99</td>
<td></td>
</tr>
<tr>
<td>Tomagachi</td>
<td>18.99</td>
<td></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Person</th>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dan</td>
<td>555-123-4567</td>
</tr>
<tr>
<td></td>
<td>Alvin</td>
<td>555-234-5678</td>
</tr>
<tr>
<td></td>
<td>Magda</td>
<td>555-345-6789</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orders</th>
<th>PName</th>
<th>Date</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
</tr>
<tr>
<td></td>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
</tr>
<tr>
<td></td>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th>ProdName</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Furby</td>
<td>9.99</td>
</tr>
<tr>
<td></td>
<td>Magic8</td>
<td>15.99</td>
</tr>
<tr>
<td></td>
<td>Tomagachi</td>
<td>18.99</td>
</tr>
</tbody>
</table>

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

Convert each table to a separate object/document?
## From Relational to Semi-Structured

<table>
<thead>
<tr>
<th>Person</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Phone</strong></td>
<td></td>
</tr>
<tr>
<td>Dan</td>
<td>555-123-4567</td>
<td></td>
</tr>
<tr>
<td>Alvin</td>
<td>555-234-5678</td>
<td></td>
</tr>
<tr>
<td>Magda</td>
<td>555-345-6789</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Orders</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PName</strong></td>
<td><strong>Date</strong></td>
<td><strong>Product</strong></td>
</tr>
<tr>
<td>Dan</td>
<td>1997</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2000</td>
<td>Furby</td>
</tr>
<tr>
<td>Alvin</td>
<td>2012</td>
<td>Magic8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ProdName</strong></td>
<td><strong>Price</strong></td>
<td></td>
</tr>
<tr>
<td>Furby</td>
<td>9.99</td>
<td></td>
</tr>
<tr>
<td>Magic8</td>
<td>15.99</td>
<td></td>
</tr>
<tr>
<td>Tomagachi</td>
<td>18.99</td>
<td></td>
</tr>
</tbody>
</table>

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!
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
- 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
  - Nested relational algebra: [https://dl.acm.org/citation.cfm?id=588133](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++

SQL
Demo Time!
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
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.
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 }
*/
SELECT x.phone
FROM {{
    "name": "Dan", "phone": [300, 150],
    "name": "Alvin", "phone": 420
}} AS x;

-- same output as array data
-- 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]
*/
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 }
*/
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] }
{}
*/
SELECT x.fone -- intentional typo
FROM [
    {"name": "Dan", "phone": [300, 150]},
    {"name": "Alvin", "phone": 420}
] AS x;

-- output, beware of typos!
/*@*/
SELECT x.fone -- intentional typo
FROM [
    {
        "name": "Dan",
        "phone": [300, 150]
    },
    {
        "name": "Alvin",
        "phone": 420
    }
] AS x;

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