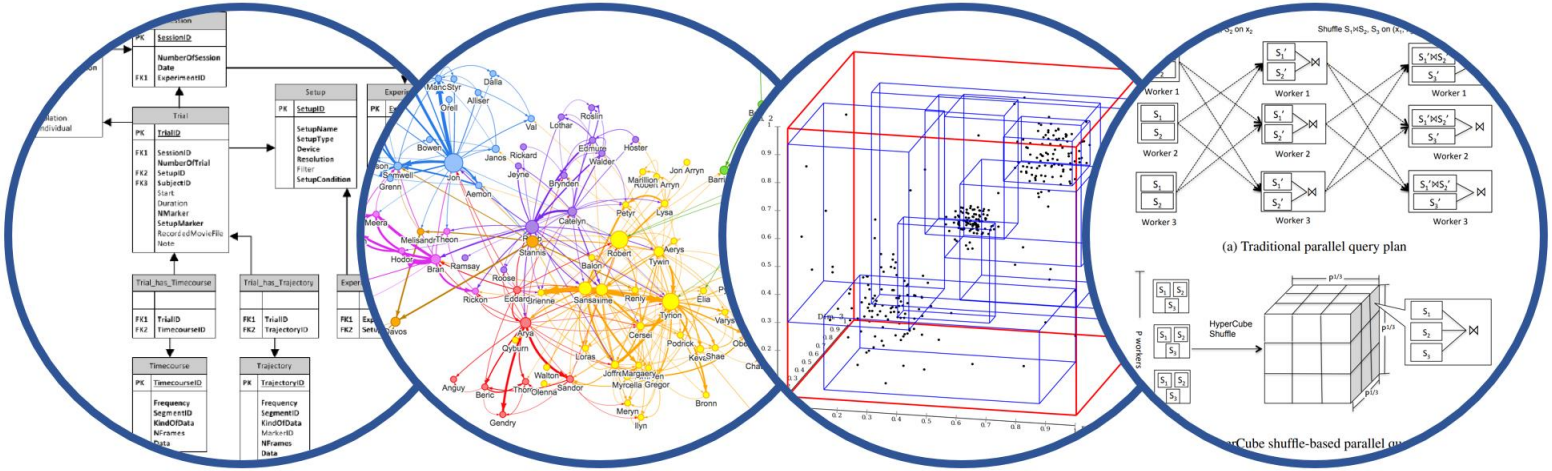


Introduction to Data Management

Semi-structured Data

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

Announcements

- This is a recording for Lecture 27, November 27

- HW6: Part 2 is due tonight!

Semistructured Data

Outline

- In the Relational Data Model all relations are in First Normal Form (1NF): they are flat relations
- Semistructured data means nested data: non-1NF
- Examples:
  - JSON data
  - Protobuf data
  - XML data
  - …

Applications

Where is Semistructured used:

- To exchange data
- To store small, static datasets
- By some NoSQL databases (next)

NoSQL Systems

Goal of NoSQL: higher TPS

- Distribute data on many servers
- Replicate data on 3-5 servers
- Data model: key,value pairs, (k,v)
- GET(k), PUT(k,v), no query language
- TXN: single-update only
- Examples: MongoDB, CouchDB, Cassandra

The value v can be a JSON file "Document Store"

Outline

This lecture

- Semi-structured data model in JSON

- Introducing AsterixDB and SQL++

Next week

Semi-Structured Documents

- Some notion of **tagging** to mark down semantics
- Examples:
  - **XML**
  - Protobuf
  - 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>
```

Tags surround the respective data

Semi-Structured Documents

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

```
message SearchRequest {
  string query = 1;
  int32 page_number = 2;
  int32 results_per_page = 3;
}
```

https://protobuf.dev/programming-guides/editions/

Semi-Structured Documents

- Some notion of **tagging** to mark down semantics
- Examples:
  - XML
  - Protobuf
  - **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"
                }
            ]
        }
    ]
}
```

Tags introduce the respective data

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": null,
         "sale": true
      }
   ]
}
```

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": null,
         "sale": true
      }
   ]
}
```

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": null,
         "sale": true
      }
   ]
}
```

Types

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

- "name": <value>
- Values can be primitives, objects, or arrays
- 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": null,
            "sale": true
        }
    ]
}
```

Types

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

- "name": <value>
- Values can be primitives, objects, or arrays
- 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": null,
         "sale": true
      }
   ]
}
```

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

NOT ALLOWED
(duplicated authors)

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

OK
(author array)

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

Thinking About Semi-Structured Data

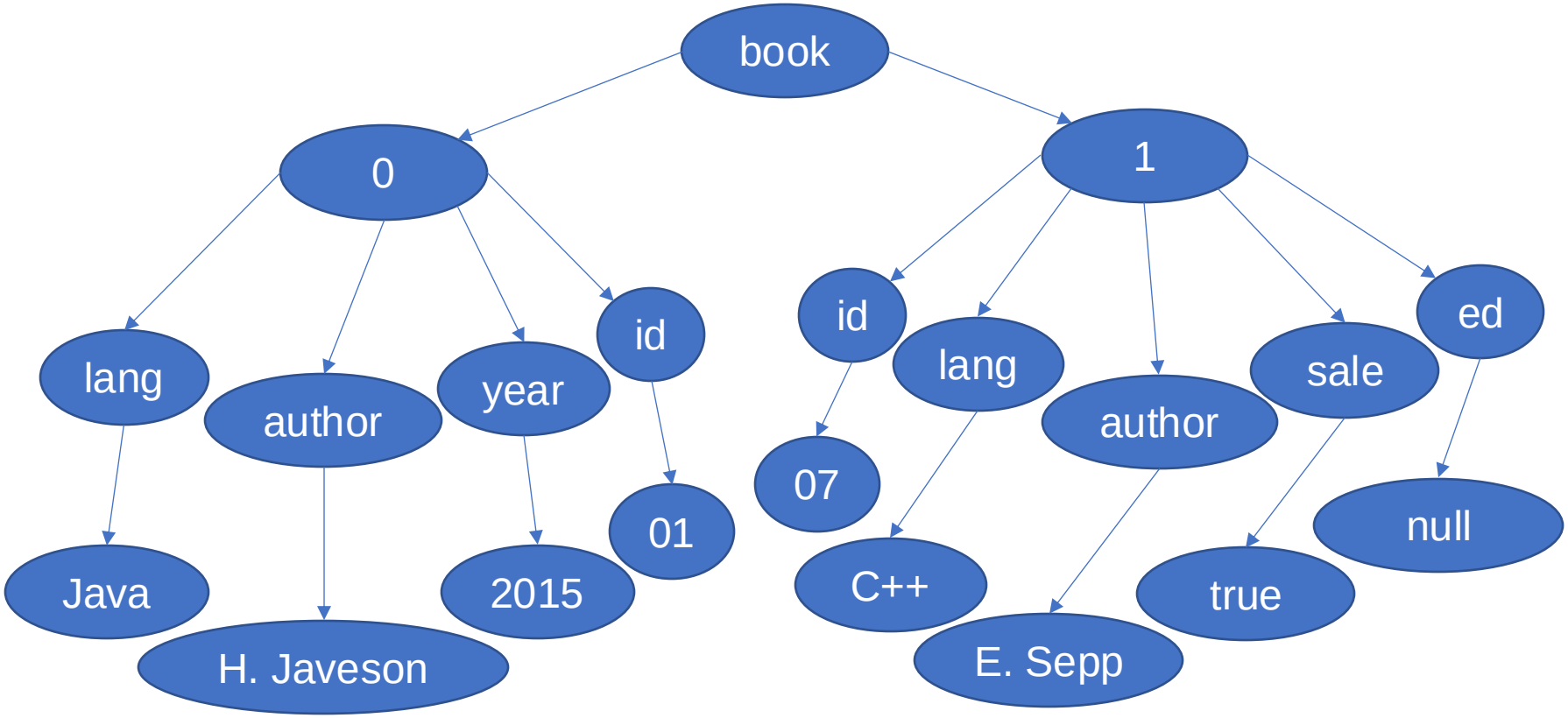
What does semi-structured data structure encode?

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

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?

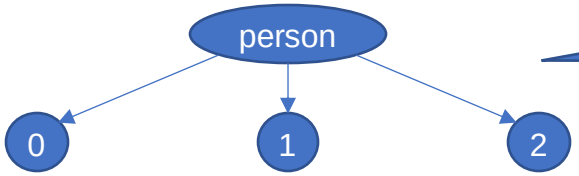
person

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?



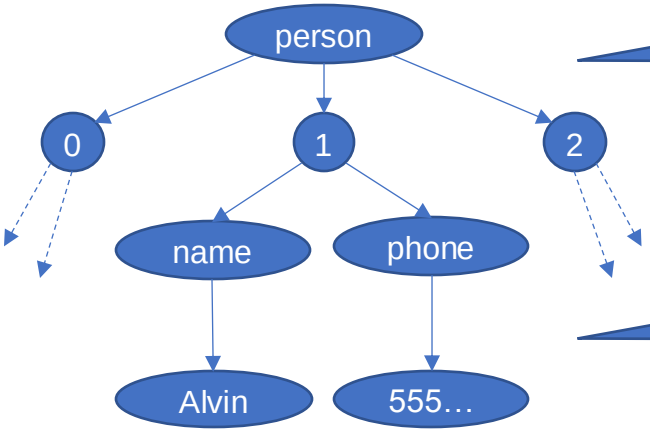
Tables are just an array of elements (rows)

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?



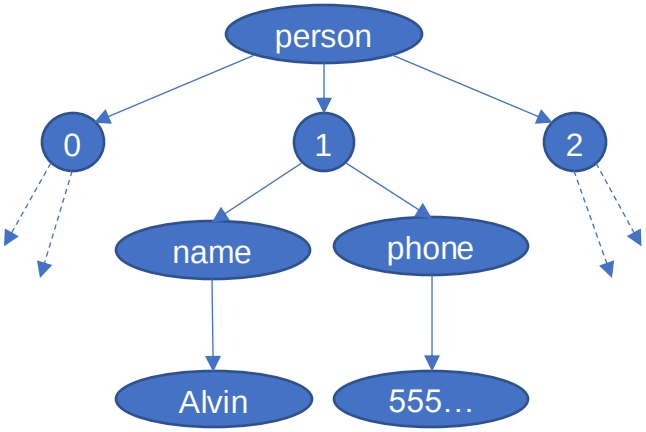
Tables are just an array of elements (rows)

Rows are just simple (unnested) 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

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": "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?

Nested 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?

Nested 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?

Nested 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

Representing a
one-to-many relationship

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

Representing a one-to-many relationship

```
{
   "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

BCNF

Orders

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

Representing a one-to-many relationship

```
{
   "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": []
      }
   ]
}
```

Unnormalized data

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

Representing a many-to-many relationship is more difficult

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

Representing a many-to-many relationship is more difficult

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

Representing a many-to-many relationship is more difficult

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

Representing a many-to-many relationship is more difficult

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

Representing a many-to-many relationship is more difficult

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

Representing a many-to-many relationship is more difficult

Convert each table to a separate array/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

Representing a many-to-many relationship is more difficult

Convert each table to a separate array/document?

We wanted to avoid joining in the first place!

Summary of Semi-structured data

- Data model = a tree
- Text file: parsed, self-describing data
- Unnormalized: Non-1NF, Non-BCNF
- Easy to represent 1-to-many relationships
- Many-to-many relationships lead to redundancies