Introduction to Data Management

Key-Value vs Semi-Structured Data

Alyssa Pittman
Based on slides by Jonathan Leang, Dan Suciu, et al

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

Do sign up for AWS Educate early! Some people have had to verify their student info before receiving their accounts.
Recap: NoSQL in a Nutshell

- NoSQL → Looser data model
  - Give up built-in OLAP/analysis functionality
  - Give up built-in ACID consistency
Outline

- KV Store
  - Hash Table (Key → Blob)

- Document Store
  - Hash Table + Parsable Documents
NoSQL Data Models

Key-Value Database

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>AAA,BBB,CCC</td>
</tr>
<tr>
<td>K2</td>
<td>AAA,BBB</td>
</tr>
<tr>
<td>K3</td>
<td>AAA,DDD</td>
</tr>
<tr>
<td>K4</td>
<td>AAA,2,01/01/2015</td>
</tr>
<tr>
<td>K5</td>
<td>3,ZZZ,5623</td>
</tr>
</tbody>
</table>

Wide-Column Store (Extensible Record Store)

Graph Database

Document Store

XML

```
<empinfo>
    <employees>
        <employee>
            <name>James Kirk</name>
            <age>40</age>
        </employee>
        <employee>
            <name>Jean-Luc Picard</name>
            <age>45</age>
        </employee>
        <employee>
            <name>Wesley Crusher</name>
            <age>27</age>
        </employee>
    </employees>
</empinfo>
```

JSON

```
{
    "empinfo": {
        "employees": [
            {
                "name": "James Kirk",
                "age": 40,
            },
            {
                "name": "Jean-Luc Picard",
                "age": 45,
            },
            {
                "name": "Wesley Crusher",
                "age": 27,
            }
        ]
    }
}
```
NoSQL Data Models

Key-Value Database

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>AAA, BBB, CCC</td>
</tr>
<tr>
<td>K2</td>
<td>AAA, BBB</td>
</tr>
<tr>
<td>K3</td>
<td>AAA, DDD</td>
</tr>
<tr>
<td>K4</td>
<td>AAA, 2,01/01/2015</td>
</tr>
<tr>
<td>K5</td>
<td>3, ZZZ, 5623</td>
</tr>
</tbody>
</table>

Wide-Column Store (Extensible Record Store)

Graph Database

Document Store

XML

```xml
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```json
{
  "empinfo": {
    "employees": [
      {
        "name": "James Kirk",
        "age": 40,
      },
      {
        "name": "Jean-Luc Picard",
        "age": 45,
      },
      {
        "name": "Wesley Crusher",
        "age": 27,
      }
    ]
  }
}
```
NoSQL Data Models

Key-Value Database

- Key to value pairs
- “A hash table”

Wide-Column Store
(Extensible Record Store)

Graph Database

Document Store

XML

```xml
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```json
{
  "empinfo": {
    "employees": [
      {
        "name": "James Kirk",
        "age": 40
      },
      {
        "name": "Jean-Luc Picard",
        "age": 45
      },
      {
        "name": "Wesley Crusher",
        "age": 27
      }
    ]
  }
}
```
NoSQL Data Models

Key-Value Database

- amazon DynamoDB
- RocksDB
- redis

Wide-Column Store (Extensible Record Store)

Graph Database

Document Store

XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```
{
  "empinfo": {
    "employees": [
      {
        "name": "James Kirk",
        "age": 40,
      },
      {
        "name": "Jean-Luc Picard",
        "age": 45,
      },
      {
        "name": "Wesley Crusher",
        "age": 27,
      }
    ]
  }
}
```
NoSQL Data Models

Key-Value Database

- Amazon DynamoDB
- RocksDB
- Redis

Graph Database

Persistent KV Store

Wide-Column Store (Extensible Record Store)

Document Store

XML

```
<emplinfo>  
  <employees>  
    <employee>  
      <name>James Kirk</name>  
      <age>40</age>  
    </employee>  
    <employee>  
      <name>Jean-Luc Picard</name>  
      <age>45</age>  
    </employee>  
    <employee>  
      <name>Wesley Crusher</name>  
      <age>27</age>  
    </employee>  
  </employees>  
</emplinfo>
```

JSON

```
{  
  "emplinfo":  
    {  
      "employees":  
        [  
          {  
            "name": "James Kirk",  
            "age": 40,  
          },  
          {  
            "name": "Jean-Luc Picard",  
            "age": 45,  
          },  
          {  
            "name": "Wesley Crusher",  
            "age": 27,  
          }  
        ]  
    }  
}
NoSQL Data Models

Key-Value Database

- Amazon DynamoDB
- RocksDB
- Redis

Wide-Column Store (Extensible Record Store)

In-memory KV store

Graph Database

Persistent KV store

Document Store

XML

```xml
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```json
{
  "empinfo": {
    "employees": [
      {
        "name": "James Kirk",
        "age": 40
      },
      {
        "name": "Jean-Luc Picard",
        "age": 45
      },
      {
        "name": "Wesley Crusher",
        "age": 27
      }
    ]
  }
}
```
NoSQL Data Models

Key-Value Database
- Amazon DynamoDB
- RocksDB
- Redis

Extended to also be a Document Store

In-memory KV store

Graph Database

Document Store

Extended to also be a Document Store

In-memory KV store

XML

```xml
<eminfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</eminfo>
```

JSON

```json
{
  "eminfo": {
    "employees": [
      {
        "name": "James Kirk",
        "age": 40,
      },
      {
        "name": "Jean-Luc Picard",
        "age": 45,
      },
      {
        "name": "Wesley Crusher",
        "age": 27,
      }
    ]
  }
}
```
Key-Value Store

- Data model:
  - (key, value) pairs
  - Key 🡪 string/integer/…, unique for the entire data
  - Value 🡪 anything
Key-Value Store

- **Data model:**
  - (key, value) pairs
  - Key □ string/integer/..., unique for the entire data
  - Value □ anything

- **Basic Operations:**
  - get(key)
  - put(key, value)
Key-Value Store

- **Data model:**
  - `(key, value)` pairs
  - Key → string/integer/..., unique for the entire data
  - Value → anything

- **Basic Operations:**
  - `get(key)`
  - `put(key, value)`

- **Distribution/Partitioning:**
  - Access via hash function
  - No replication: Key `k` stored at server `h(k)\%N`
  - 3-way replication: Key `k` stored at servers `h_1(k)\%N, h_2(k)\%N, h_3(k)\%N`
Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
</table>

Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>Single flight record</td>
</tr>
</tbody>
</table>
Key-Value Modeling

Represent all Flights as KV pairs

Potential KV pairings

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>Single flight record</td>
</tr>
<tr>
<td>Date</td>
<td>All flight records on that day</td>
</tr>
</tbody>
</table>
# Key-Value Modeling

**Represent all Flights as KV pairs**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FID</td>
<td>Single flight record</td>
</tr>
<tr>
<td>Date</td>
<td>All flight records on that day</td>
</tr>
<tr>
<td>(origin, destination)</td>
<td>All flight records between the cities</td>
</tr>
</tbody>
</table>
DynamoDB API

- Create, Read, Update, Delete (CRUD) actions
  - Create 🡪 **PutItem**
  - Read 🡪 **GetItem**
  - Update 🡪 **UpdateItem** (Document store functionality)
  - Delete 🡪 **DeleteItem**

- Read consistency
  - Eventually consistent (default, may be stale data)
  - Strongly consistent (gets most recent written data)

- As of December 2018, ACID is “supported”
  - **TransactWriteItems**
  - **TransactGetItems**
NoSQL Data Models

**Key-Value Database**

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>AAA, BBB, CCC</td>
</tr>
<tr>
<td>K2</td>
<td>AAA, BBB</td>
</tr>
<tr>
<td>K3</td>
<td>AAA, DDD</td>
</tr>
<tr>
<td>K4</td>
<td>AAA, 201/01/2015</td>
</tr>
<tr>
<td>K5</td>
<td>3, ZZZ, 5623</td>
</tr>
</tbody>
</table>

**Wide-Column Store**

(Extensible Record Store)

**Graph Database**

**Document Store**

XML

```
<empinfo>
  <employees>
    <employee>
      <name>James Kirk</name>
      <age>40</age>
    </employee>
    <employee>
      <name>Jean-Luc Picard</name>
      <age>45</age>
    </employee>
    <employee>
      <name>Wesley Crusher</name>
      <age>27</age>
    </employee>
  </employees>
</empinfo>
```

JSON

```json
{  
  "empinfo" : {  
    "employees" : [  
      { "name" : "James Kirk",  
        "age" : 40,  
      },  
      { "name" : "Jean-Luc Picard",  
        "age" : 45,  
      },  
      { "name" : "Wesley Crusher",  
        "age" : 27,  
      }  
    ]  
  }  
}
```
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
  - 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

Not human readable in serialized format
Semi-Structured Documents

- Some notion of **tagging** to mark down semantics

- Examples:
  - XML
  - Protobuf
  - JSON

Tags introduce the respective data
Semi-Structured Documents

- Some notion of tagging to mark down semantics

- Examples:
  - XML
  - Protobuf
  - JSON

Many applications have phased out XML in favor of JSON

Tags introduce the respective data
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

- Basic retrieval process:
  1. Retrieve table
  2. Run through rows
  3. Return 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. Retrieve table
  2. Run through rows
  3. Return data

- Basic retrieval process:
  1. Retrieve document
  2. Parse document tree
  3. Return data

Inefficient encoding / Easy exchange of data
• 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": null,
      "sale": true
    }
  ]
}
```
**JSON Standard – Rules of the Game**

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

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

**Primitives** include:
- String (in quotes)
- Numeric (unquoted number)
- Boolean (unquoted true/false)
- Null (literally just null)
JavaScript Object Notation (JSON)

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

### Objects

- **Objects** are an *unordered* collection of name-value pairs:
  - "name": <value>
  - Values can be primitives, objects, or arrays
  - Enclosed by `{ }`

```json
{
    "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"

Types

Objects are an unordered collection of name-value pairs:
- "name": <value>
- Values can be primitives, objects, or arrays
- Enclosed by { }

```json
{
  "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"

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

**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. Javeson",
  "author": "D.Suciu",
  "author": "A. Cheung",
  "year": 2015
}
```
- 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
}
```
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": null,
      "sale": true
    }
  ]
}
```
Thinking About Semi-Structured Data

What does semi-structured data structure encode?

**Tree semantics!**

![Diagram of tree semantics]

- **book**
  - **obj0**
    - **id**: 01
    - **year**: 2015
    - **lang**: Java
    - **author**: H. Javeson
  - **obj1**
    - **id**: 07
    - **lang**: C++
    - **author**: E. Sepp
    - **ed**: null
Thinking About Semi-Structured Data

What does semi-structured data structure encode?

Tree semantics!

These objects don’t have labels, as they are in an array.

- Book
  - obj0
    - id
    - year
    - author
      - lang
      - Java
      - H. Javeson
    - 2015
  - obj1
    - id
    - lang
    - C++
    - E. Sepp
    - ed
    - sale
    - true
    - null
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>
What is a table in semi-structured land?

Tables are just an array of elements (rows)

<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>
What is a table in semi-structured land?

Tables are just an array of elements (rows)

Rows are just simple (unnested) objects
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>

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

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"
        }
    ]
}
```
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>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"  
        }  
    ]
}
```
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>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": null
    }
  ]
}
```
How can NULL be represented?

```json
{
    "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?

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

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

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

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>

```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

Person

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dan</td>
<td>???</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
- Heterogeneous collections

```
{
  "person": [
    {
      "name": "Dan",
      "phone": [
        "555-123-4567",
        "555-987-6543"
      ]
    },
    {
      "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**
• Heterogeneous collections

```json
{  "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 Data

<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>???</td>
</tr>
</tbody>
</table>

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

Non-flat data!
- Array data
- Multi-part data
- **Heterogeneous collections**
## 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>

How do we represent foreign keys?
### 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>

```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": []
    }
  ]
}
```
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>

Precomputed equijoin!
### 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?
Person → Orders → Product
### 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?

Person -> Orders -> Product

We might miss some products!

Product data will be duplicated!
### 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 Data

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

<table>
<thead>
<tr>
<th>Orders</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PName</td>
<td>Date</td>
<td>Product</td>
<td></td>
</tr>
<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></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ProdName</td>
<td>Price</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Furby</td>
<td>9.99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Magic8</td>
<td>15.99</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tomagachi</td>
<td>18.99</td>
<td></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!
### 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? Convert each table to a separate array/document?
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?

Convert each table to a separate array/document?

We wanted to avoid joining in the first place!
Big ideas:

▪ Semi-structured data is **parsed**
  • Data model flexibility
  • Potentially lots of redundancy

▪ Semi-structured data expresses **unique patterns**
  • Collection/multi-part data
  • Precompute joins

▪ Semi-structured data **has limits**
  • Relies on relational-like patterns in some situations
Next time

- AsterixDB as a case study of Document Store
  - Introducing AsterixDB and SQL++