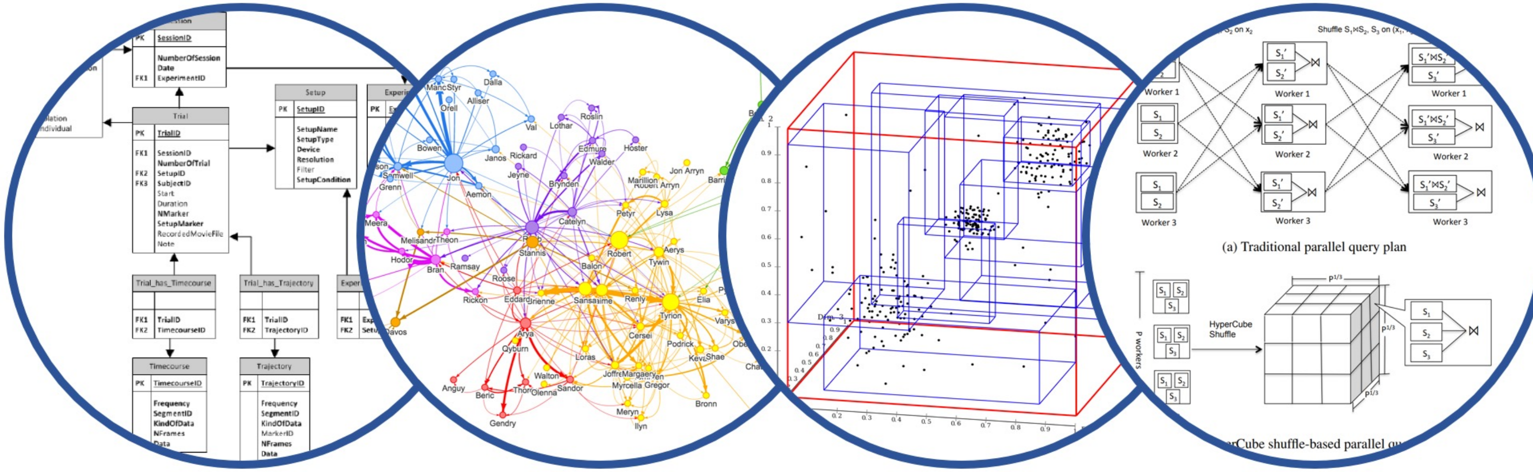


Introduction to Data Management
Semi-structured Data

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

Announcements

- **No in-person Lecture on Friday, May 24**: Please watch recorded video instead (on canvas
- TA's working hard to release feedback on HW6/1
- HW6/2 due on Friday
- HW7 to be released on Friday, due on May 31

Course Evals

- Please take a few minutes before we start to fill out the course evals

- I read every word of your comments and make adjustments where needed based on the feedback; have done this in the past

Cardinality Estimation (review+histograms)

Overview

- DBMS keeps stats for each relation R(A,B,C,…):
  - Cardinality of R: T(R)
  - Number of distinct values in R.A: V(R,A)
  - Similarly: V(R, B), V(R, C), …
  - Histograms (later)

- Cardinality estimation:
  Given only these stats, estimate output size.

Cardinality Estimation

Recursively on the query plan

- For a base table: Est(R) = T(R)

- For an operator:
  $Est(\sigma_{pred}(R)) = \theta_{pred} * Est(R)$
  $Est(R \bowtie_{A=B} S) = \theta_{A=B} * Est(R) * Est(S)$

θ is called the *selectivity factor*

Selectivity Factor

For $\sigma_{A=\text{const}}$ the selectivity factor is $\theta = \frac{1}{V(R,A)}$

Uniformity assumption

Selectivity Factor

For $\sigma_{A=const}$ the selectivity factor is $\theta = \frac{1}{V(R,A)}$

Uniformity assumption

T(Payroll) = 10000
V(Payroll, job) = 20

$\text{EST}\left[\sigma_{job=`TA'}(\text{Payroll})\right] =$??

Selectivity Factor

For $\sigma_{A=const}$ the selectivity factor is $\theta = \frac{1}{V(R,A)}$

Uniformity assumption

T(Payroll) = 10000
V(Payroll, job) = 20

$$\text{EST}\left[\sigma_{job='TA'}(\text{Payroll})\right] = \frac{1}{20} 10000 = 500$$

Selectivity Factor

For $\sigma_{\text{p and q}}$ the sel. factor is $\theta_{\text{p and q}} = \theta_p \times \theta_q$

Independence assumption

Selectivity Factor

For $\sigma_{\text{p and q}}$ the sel. factor is $\theta_{\text{p and q}} = \theta_p \times \theta_q$

Independence assumption

T(Payroll) = 10000
V(Payroll, job) = 20
V(Payroll, salary) = 50

$\text{EST}\left[\sigma_{\text{job='TA' and salary}=20000}(\text{Payroll})\right] =$??

Selectivity Factor

For $\sigma_{\text{p and q}}$ the sel. factor is $\theta_{\text{p and q}} = \theta_p \times \theta_q$

Independence assumption

T(Payroll) = 10000
V(Payroll, job) = 20
V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{job='TA' and salary=20000}}(\text{Payroll})\right] = \frac{1}{20}\frac{1}{50}10000 = 10$$

Selectivity Factor

$R \bowtie_{A=B} S$ the sel factor is $\theta = \frac{1}{\max(V(R,A),V(S,B))}$

Containment of values assumption

Selectivity Factor

$R \bowtie_{A=B} S$ the sel factor is $\theta = \dfrac{1}{\max(V(R,A), V(S,B))}$

Containment of values assumption

Justification:

- If $V(R,A) \leq V(S,B)$ then assume $R.A \subseteq S.B$
- 1 tuple in R joins $\frac{1}{V(S,B)} T(S)$ tuples in S
- Total output: $\frac{1}{V(S,B)} T(R) T(S)$

Selectivity Factor

$R \bowtie_{A=B} S$ the sel factor is $\theta = \frac{1}{\max(V(R,A),V(S,B))}$

Containment of values assumption

Justification:

- If $V(R,A) \leq V(S,B)$ then assume $R.A \subseteq S.B$
- 1 tuple in R joins $\frac{1}{V(S,B)} T(S)$ tuples in S
- Total output: $\frac{1}{V(S,B)} T(R) T(S)$

T(Payroll) = 10000
V(Payroll, UserID) = 10000

T(Regist) = 3000
V(Regist, UserID) = 2000

$EST[Payroll \bowtie Regist] =$??

Selectivity Factor

$R \bowtie_{A=B} S$ the sel factor is $\theta = \dfrac{1}{\max(V(R,A),V(S,B))}$

Containment of values assumption

Justification:

- If $V(R,A) \leq V(S,B)$ then assume $R.A \subseteq S.B$
- 1 tuple in R joins $\frac{1}{V(S,B)} T(S)$ tuples in S
- Total output: $\frac{1}{V(S,B)} T(R) T(S)$

T(Payroll) = 10000
V(Payroll, UserID) = 10000

T(Regist) = 3000
V(Regist, UserID) = 2000

$$\text{EST}[\text{Payroll} \bowtie \text{Regist}] = \frac{1}{\max(10000,2000)} 10000 \cdot 3000 = 3000$$

Summary of Assumptions

- Uniformity
- Independence
- Containment of values
- Preservation of values

Computing the Cost of a Plan

- Estimate cardinalities bottom-up
- Estimate cost by using estimated cardinalities

Histograms

- Histogram on R.A refines T(R), V(R,A)
- Each bucket contains: T(bucket), V(bucket, A)

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\mathrm{EST}\left[\sigma_{\mathrm{salary}=25\mathrm{k}}(\mathrm{Payroll})\right] = ???$$

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = ???$$

$$= \frac{1}{50} 10000 = 10$$

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = ??? \qquad = \frac{1}{50} 10000 = 10$$

Salary:	0..20k	20k..29k	30k-39k	40k-49k	50k-59k	> 60k
T =	80	320	2000	4800	2600	200
V =	5	10	10	15	5	5

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{salary=25k}(\text{Payroll})\right] = ???$$

$$= \frac{1}{50} 10000 = 10$$

Salary:	0..20k	20k..29k	30k-39k	40k-49k	50k-59k	> 60k
T =	80	320	2000	4800	2600	200
V =	5	10	10	15	5	5

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = ???$$

$$= \frac{1}{50} 10000 = 10$$

Salary:	0..20k	20k..29k	30k-39k	40k-49k	50k-59k	> 60k
T =	80	320	2000	4800	2600	200
V =	5	10	10	15	5	5

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = \frac{1}{10} 320 = 32$$

Histograms

T(Payroll) = 10000, V(Payroll, salary) = 50

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = ???$$

$$= \frac{1}{50} 10000 = 10$$

Salary:	0..20k	20k..29k	30k-39k	40k-49k	50k-59k	> 60k
T =	80	320	2000	4800	2600	200
V =	5	10	10	15	5	5

$$\text{EST}\left[\sigma_{\text{salary}=25\text{k}}(\text{Payroll})\right] = \frac{1}{10} 320 = 32$$

A better estimate

Recap: How a Query Engine Works

- Each RA operator has multiple physical operators
- Indexes: speed up some queries slow down others
- Rewrite rules transform one query plan to another
- Cardinality estimators used to estimate the cost; they are often wrong

Semistructured Data and SQL++

NoSQL Systems

RDBMS: Transactions Per Second (TPS) < 10k

NoSQL:

NoSQL Systems

RDBMS: Transactions Per Second (TPS) < 10k

NoSQL:

- Distribute data on multiple servers

Increase throughput

NoSQL Systems

RDBMS: Transactions Per Second (TPS) < 10k

NoSQL:

- Distribute data on multiple servers
- Replicate each data item on 3-5 servers

Increase throughput

No need to store on disk

NoSQL Systems

RDBMS: Transactions Per Second (TPS) < 10k

NoSQL:

- Distribute data on multiple servers
- Replicate each data item on 3-5 servers
- Simple data model: key,value pairs
- API: GET(k), PUT(k,v)

Increase throughput

No need to store on disk

NoSQL Systems

RDBMS: Transactions Per Second (TPS) < 10k

NoSQL:

- Distribute data on multiple servers
- Replicate each data item on 3-5 servers
- Simple data model: key,value pairs
- API: GET(k), PUT(k,v)
- No full query language
- Single-update transactions only

Increase throughput

No need to store on disk

NoSQL Systems

RDBMS: Transactions Per Second (TPS) < 10k

NoSQL:

- Distribute data on multiple servers
- Replicate each data item on 3-5 servers
- Simple data model: key,value pairs
- API: GET(k), PUT(k,v)
- No full query language
- Single-update transactions only
- Examples: MongoDB, CouchDB, Cassandra

Increase throughput

No need to store on disk

NoSQL Systems

RDBMS: Transactions Per Second (TPS) < 10k

NoSQL:

- Distribute data on multiple servers
- Replicate each data item on 3-5 servers
- Simple data model: key,value pairs
- API: GET(k), PUT(k,v)
- No full query language
- Single-update transactions only
- Examples: MongoDB, CouchDB, Cassandra

Increase throughput

No need to store on disk

But what is the value v? Often: v = a document

Outline

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

Outline

- 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

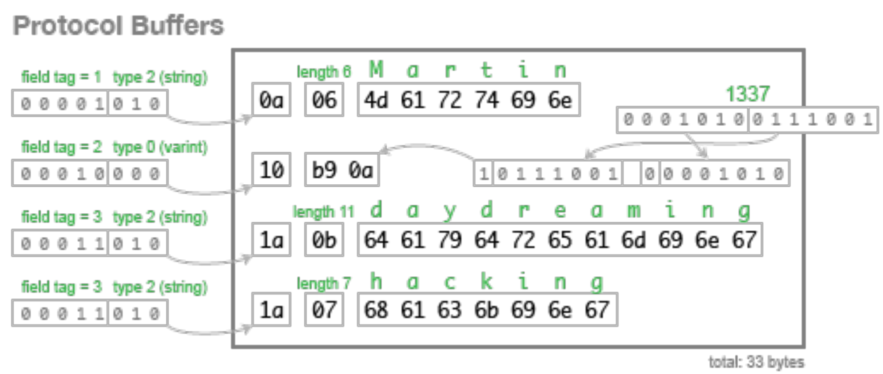
- Some notion of **tagging** to mark down semantics
- Examples:
  - **XML**
  - Protobuf
  - JSON

```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



Able to record field number and type but not name

Semi-Structured Documents

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


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

Less well-defined/More flexible

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

- Semi-Structured
  - Basic retrieval process:
    1. Retrieve document
    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. 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

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

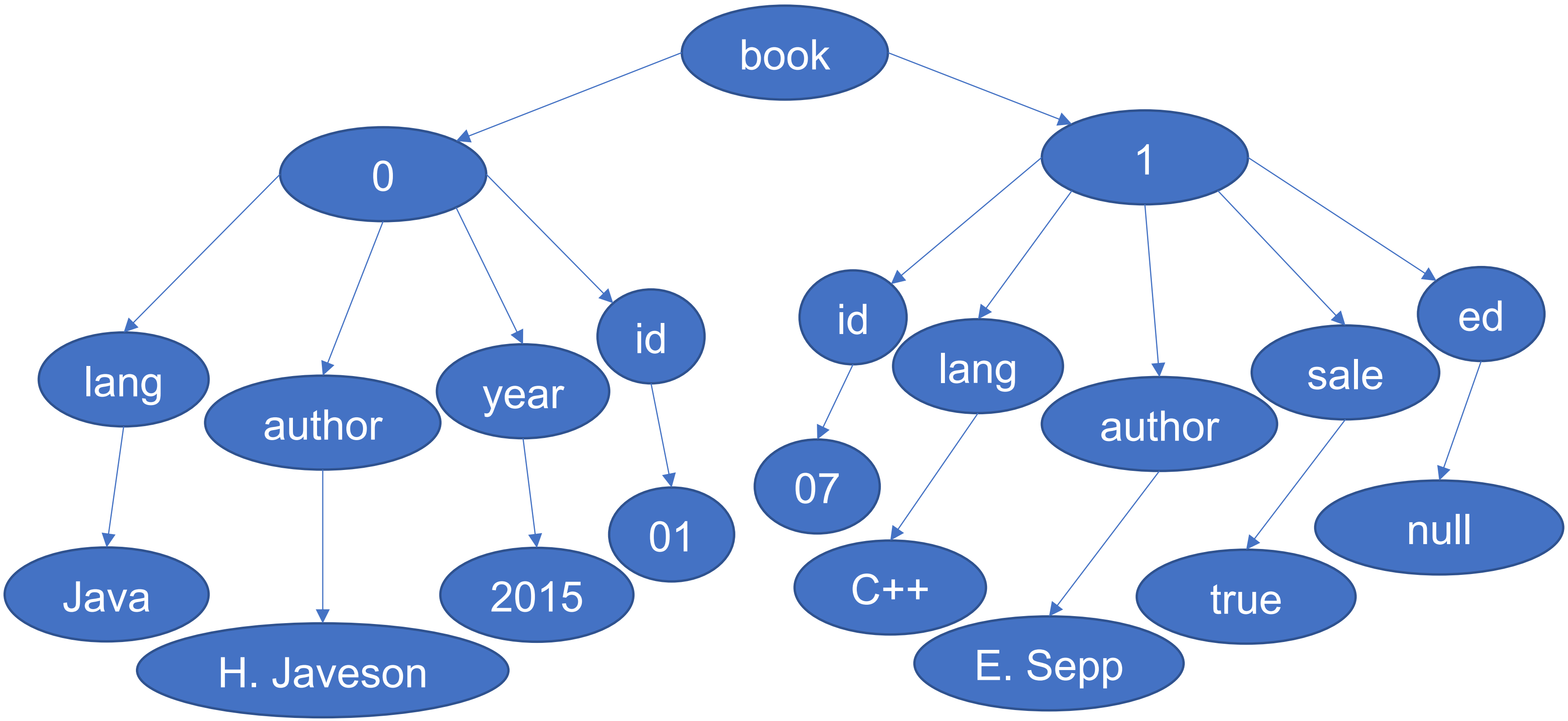
What does semi-structured data structure encode?

```
{
   "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

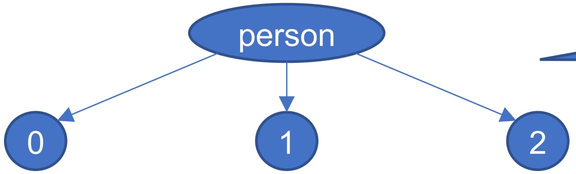
What is a table in semi-structured land?

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?



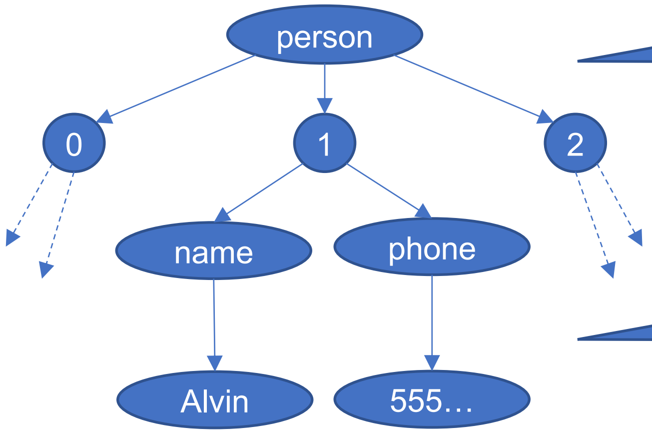
Tables are just an array of elements (rows)

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?



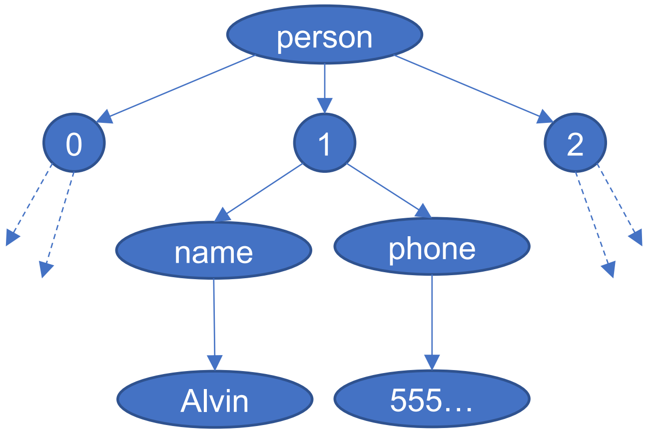
Tables are just an array of elements (rows)

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?

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 in one big file??

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 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

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!

From Relational to Semi-Structured

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