Ghislain Fourny

Big Data

11. Document stores
The semi-structured stack so far
The semi-structured stack so far
The semi-structured stack so far

Well-formed XML/JSON

Text

Bits
The semi-structured stack so far

Valid XML/JSON
Well-formed XML/JSON
Text
Bits
The semi-structured stack so far

Queryable XML/JSON
Valid XML/JSON
Well-formed XML/JSON
Text
Bits
Encoding
Encoding

Character

0s and 1s
Common Character Encodings

<table>
<thead>
<tr>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASCII</td>
</tr>
<tr>
<td>ISO Latin 1</td>
</tr>
<tr>
<td>(a.k.a. ISO-8859-1)</td>
</tr>
<tr>
<td>UTF-8</td>
</tr>
<tr>
<td>UTF-16</td>
</tr>
</tbody>
</table>
The ASCII code chart is shown in the image. The chart lists various ASCII characters and their corresponding code values. The chart includes entries for NUL (0), DLE (1), SP (2), O (3), @ (4), P (5), \ (6), and DEL (7). The chart also includes rows for SOH, DC1, DC2, ETX, DC3, EOT, DC4, ENQ, ACK, SYN, BEL, BS, CAN, HT, LF, SUB, VT, FF, CR, SO, RS, SI, US, and DEL.

ASCII Code Chart, scanner copied from the material delivered with TermiNet 300 impact type printer with Keyboard, February 1972, General Electric Data communication Product Dept., Waynesboro VA.  
UTF-8

π

03A0

11 10100000

11001110 10100000
BSON

```
{ "foo" : null }
```

6 \x03 \x66 \x6F \x6F \x00 \x0A \x00
Mappings
Relational table

```json
{
  "foo": 1,
  "bar": "foo"
}
{
  "foo": 2,
  "bar": "bar"
}
{
  "foo": 3,
  "bar": "foobar"
}
```

<table>
<thead>
<tr>
<th>foo</th>
<th>bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>foo</td>
</tr>
<tr>
<td>2</td>
<td>bar</td>
</tr>
<tr>
<td>3</td>
<td>foobar</td>
</tr>
</tbody>
</table>
Relational table

```json
{
  "foo": 1,
  "bar": "foo",
  "name": {
    "last": "Einstein",
    "first": "Albert"
  }
}
{
  "foo": 2,
  "bar": "bar"
}
{
  "foo": 3,
  "name": {
    "last": "Gödel",
    "first": "Kurt"
  }
}
```

<table>
<thead>
<tr>
<th>foo</th>
<th>bar</th>
<th>name. last</th>
<th>name. first</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>foo</td>
<td>Einstein</td>
<td>Albert</td>
</tr>
<tr>
<td>2</td>
<td>bar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>Gödel</td>
<td>Kurt</td>
</tr>
</tbody>
</table>
Common approaches

Schema-based Shredding

Edge Shredding

Tree Encoding
# Schema-based Shredding

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>ZRH</td>
<td>Zurich</td>
<td>150</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>PAR</td>
<td>Paris</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>PAR</td>
<td>Paris</td>
<td>100</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>LHR</td>
<td>London Heathrow</td>
<td>100</td>
<td>2</td>
</tr>
</tbody>
</table>
# Edge Shredding

<table>
<thead>
<tr>
<th>Ordinal</th>
<th>Source</th>
<th>Label</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Doc</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>Airport</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>airId</td>
<td>v1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>name</td>
<td>v2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>tax</td>
<td>v3</td>
</tr>
</tbody>
</table>

```xml
<doc>
  <Airport airId="ZRH">
    <name>Zurich</name>
    <tax>150</tax>
  </Airport>
  ...
</doc>
```

<table>
<thead>
<tr>
<th>Id</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1</td>
<td>ZRH</td>
</tr>
<tr>
<td>v2</td>
<td>Zurich</td>
</tr>
<tr>
<td>v3</td>
<td>150</td>
</tr>
</tbody>
</table>
Tree encoding

<table>
<thead>
<tr>
<th>pre</th>
<th>size</th>
<th>level</th>
<th>kind</th>
<th>prop</th>
<th>frag</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>(many)</td>
<td>1</td>
<td>element</td>
<td>Doc</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>2</td>
<td>element</td>
<td>Airport</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>attribute</td>
<td>airId</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>4</td>
<td>attvalue</td>
<td>ZRH</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
<td>element</td>
<td>name</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>4</td>
<td>text</td>
<td>Zurich</td>
<td></td>
</tr>
</tbody>
</table>

```
<doc>
  <Airport airId="ZRH">
    <name>Zurich</name>
    <tax>150</tax>
  </Airport>

...  
</doc>```
Impedance mismatch
IM1: Tree vs. flat
IM2: Schema vs. Self-describing
Document stores
Document Stores

Scale up: millions to billions

(TB, PB)
Implementations
Document Stores vs. RDBMS

+ Projection

+ Selection

- Joins
Data Denormalization

3NF

0NF
CRUD

Create
Read
Update
Delete
Read

db.scientists.find(
    { "Theory" : "Particle Physics" } )
Read: projection

db.scientists.find(
    { "Theory" : "Particle Physics" },
    { "Name" : 1, "BirthDate": 1 }
)
Read: selecting all documents

db.scientists.find({})

db.scientists.find()
Read: navigation

db.scientists.find({
  "Name.First" : "Albert"
})

matches

{
  "Name" : {
    "First" : "Albert",
    "Last" : "Einstein"
  },
  "Theories": [ "Relativity" ]
}
Read: possible confusion

db.scientists.find({
    "Name" : { "First" : "Albert" }
})

Will attempt to find an exact match at the object level!

{ "Name" : { "First" : "Albert" },
  "Theories" : [ "Relativity" ]
}
Read: matching array elements

db.scientists.find({
   "Theories" : "Special relativity"
})

Will match any element (existential quantifier)

{  
   "Name" : {
      "First" : "Albert"
   },
   "Theories": [
      "Special relativity",
      "General relativity"
   ]
}
Read: operators

db.scientists.find({
   "Number of publications" : { "$gte" : 100 } })

Also: $eq, $ne, $gt, $lt, $lte
Read: operators

db.scientists.find({
    "University" : {
        "$in" : [ "ETH Zurich", "EPFL" ]
    }
})
Writing: atomicity

Granularity of atomicity: one document
Insert

db.flights.insert(
    { "Name" : "Einstein", "Theory" : "Relativity" } 
)
Update

db.scientists.update(
    { "Name" : "Einstein" },
    { "$set" : { "Century" : "20" } }
)
Remove

db.scientists.remove(
  { "century" : "15" }
)
Query document stores on a higher level?

...it's on the way

<\textit{XQuery/}> \textit{UNQL} \{\textit{JSONiq}\}
Principle 8. Shard the data
Principle 9. Replicate the data
Principle 10. Buy lots of cheap hardware
Replication in document stores

<table>
<thead>
<tr>
<th>Master</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Clustering in document stores
Clustering
Replica sets on physical level

Replica set

Primary

Secondary

Secondary
Replica sets on physical level

Shard 1

Shard 2

Shard 3
Write concerns
Write concerns
Write concerns
Write concerns
Write concerns
Write concerns
Indices
Indices

{  
  Name: "Apple", "Color": [ "green", "red" ]
}

{  
  Name: "Orange", "Color": [ "orange" ]
}

{  
  Name: "Banana", "Color": [ "yellow" ]
}

{  
  Name: "Kiwi", "Color": [ "brown", "green" ]
}

{  
  Name: "Ananas", "Color": [ "yellow" ]
}
Indices

- **yellow**: { Name: "Apple", "Color": [ "green", "red" ] }
- **orange**: { Name: "Orange", "Color": [ "orange" ] }
- **red**: { Name: "Banana", "Color": [ "yellow" ] }
- **green**: { Name: "Kiwi", "Color": [ "brown", "green" ] }
- **brown**: { Name: "Ananas", "Color": [ "yellow" ] }
Hash indices

green

H

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tree indices (B-trees)
Indices

Primary: _id

Secondary: other fields
Query with no indices

Scan/filter in memory
Query with indices

Prefilter with index

More scan/filter in memory
Index creation: hash

db.scientists.createIndex({
  "Name.Last" : "hash"
})
Query with index

db.scientists.find({
    "Name.Last" : "Einstein"
    "Theories" : "Relativity"
})
Index creation: B-Tree

db.scientists.createIndex({
    "Name.Last" : 1
})
Index creation: compound

db.scientists.createIndex({
    "Birth date" : 1,
    "Death date" : -1
})
Index creation: prefixes are implied

```json
{
    "Birth date" : 1,
    "Death date" : -1,
    "Name.Last" : 1
}
{
    "Birth date" : 1
}
{
    "Birth date" : 1,
    "Death date" : -1
}
```
Why? In the B-Tree, prefixes are "together"
Data Denormalization

```json
{
  code: "ZRH",
  city: "Zurich"
}
{
  code: "LHR",
  city: "Heathrow"
}
{
  code: "LX100",
  departure: "ZRH",
  arrival: "LHR"
}
{
  code: "LX200",
  departure: {
    code: "LHR",
    city: "Heathrow"
  },
  arrival: {
    code: "ZRH",
    city: "Zurich"
  }
}
{
  code: "LX200",
  departure: {
    code: "LHR",
    city: "Heathrow"
  },
  arrival: {
    code: "ZRH",
    city: "Zurich"
  }
}
{
  code: "LX100",
  departure: {
    code: "ZRH",
    city: "Zurich"
  },
  arrival: {
    code: "LHR",
    city: "Heathrow"
  }
}
```

Normal Form (NF3)
Identifying nodes
Node IDs

- Integer 7
- Double 7.07
- Dewey 1.6.2.3
- ORDPATH 1.7.1

Duck Donald

Born June 9th, 1934
Duckburg, Calisota

Duck County
The mayor
Scrooge McDuck

Issued 11/25/09
Expires 11/24/19
Integer IDs

```xml
<doc>
  <Passenger>
    <name>Santa Claus</name>
    <passnumber>000112</passnumber>
    <address>Somewhere</address>
  </Passenger>
  <Reservation>
    <date>2006-12-24</date>
    <flightRef>LX124</flightRef>
    <passRef>000111</passRef>
  </Reservation>
</doc>
```
Double IDs

<?xml version="1.0" encoding="UTF-8"?>
[1.0]<doc>
  [2.0]<Passenger>
    [3.0]<name>[4.0]Santa Claus</name>
    [5.0]<passnumber>[6.0]000112</passnumber>
    [7.0]<address>[8.0]Somewhere</address>
  </Passenger>
  [9.0]<Reservation>
    [10.0]<date>[11.0]2006-12-24</date>
    [12.0]<flightRef>[13.0]LX124</flightRef>
    [14.0]<passRef>[15.0]000111</passRef>
  </Reservation>
</doc>
Dewey IDs

<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
  [1.1]<Passenger>
    [1.1.1]<name>[1.1.1.1]Santa Claus</name>
    [1.1.2]<passnumber>[1.1.2.1]000112</passnumber>
    [1.1.3]<address>[1.1.3.1]Somewhere</address>
  </Passenger>
  [1.2]<Reservation>
    [1.2.1]<date>[1.2.1.1]2006-12-24</date>
    [1.2.2]<flightRef>[1.2.2.1]LX124</flightRef>
    [1.2.3]<passRef>[1.2.3.1]000111</passRef>
  </Reservation>
</doc>
ORDPATH IDs

<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
  [1.1]<Passenger>
    [1.1.1]<name>[1.1.1.1]Santa Claus</name>
    [1.1.3]<passnumber>[1.1.3.1]000112</passnumber>
    [1.1.5]<address>[1.1.5.1]Somewhere</address>
  </Passenger>
  [1.3]<Reservation>
    [1.3.1]<date>[1.3.1.1]2006-12-24</date>
    [1.3.3]<flightRef>[1.3.3.1]LX124</flightRef>
    [1.3.5]<passRef>[1.3.5.1]000111</passRef>
  </Reservation>
</doc>
Update: ORDPATH IDs

```xml
<?xml version="1.0" encoding="UTF-8"?>
[1]<doc>
  [1.1]<Passenger>
    [1.1.1]<name>[1.1.1.1]Santa Claus</name>
    [1.1.3]<passnumber>[1.1.3.1]000112</passnumber>
    [1.1.5]<address>[1.1.5.1]Somewhere</address>
  </Passenger>
  [1.2.1]<Reservation>
    [1.2.1.1]<date>[1.2.1.1.1]2008-12-26</date>
    [1.2.1.3]<flightRef>[1.2.1.3.1]LX183</flightRef>
    [1.2.1.5]<passRef>[1.2.1.5.1]000111</passRef>
  </Reservation>
  [1.3]<Reservation>
    [1.3.1]<date>[1.3.1.1]2006-12-24</date>
    [1.3.3]<flightRef>[1.3.3.1]LX124</flightRef>
    [1.3.5]<passRef>[1.3.5.1]000111</passRef>
  </Reservation>
</doc>
```