Using Left-Right Trees for Hierarchic Data Storage

Version: 20 September 2011

Dale Chant, Roland Seidel,
Red Centre Software Pty Ltd
SSS Conference, Bristol, 2011
Abstract

- Hierarchies such as grids (Brand Image) or cubes (Brand/Statement/Rating) are levels where no levels are parallel, or, alternatively, all levels are mutually orthogonal at the origin.

- Such N-dimensional structures must presently be stored as either flat or as a SSS v2 <hierarchy>

- But if flat, then many columns, and if as hierarchy of surveys, then many files.

- For flat storage, the problem is acute on large brand lists with sparse code instantiation.

- 1,000 brands * 10 attributes * 10 rating points = 20,000 columns, even if most respondents skip or respond for only a few out of the 1,000 brands. And if 10 such questions, then 200,000 columns.

- For hierarchic storage, multiple files for simple grids and cubes is overkill, and conceptualising as a hierarchy of surveys can be counter-intuitive where the case is a single respondent.

- This proposal for the storage of such data as left-right trees (parsable by simply reading a string from the left) can hugely reduce the number of required columns.

- For fixed width, the number of columns is determined by the longest response in the record. For delimited storage, each respondent would require only as many characters as needed to record and structure just that respondent’s answer set.

- The proposed storage could also be used to store any levels structure, but at the expense of needing to duplicate the upper paths for parallel (non-orthogonal) levels.
Left Right Trees

Left-right trees are simply a way of representing data hierarchies as a strings which can be parsed from left to right.

Assign a depth delimiter to each level – eg a, b, c, d

The top-down tree node structure

store the data at each node as

This is conceptually similar to Surveycraft loops
The SSS V2 Household Data

Household 1
Terrace, East

Person 1
Fem
21-45
Soc
Bus

Person 2
Male
21-45
Work
Train
CarP

Person 3
Male
>65
Work
Train
CarD

Household, N=3

Household 2
Semi-Det, South

Person 1
Fem
46-65
Soc
CarP

Person 2
Male
<21
CarD

Person 3
Male
21-45
Soc
Bus

Household 3
Flat, East

Person 1
Fem
21-45
Soc
Bus

Person N=6

Trip, N=12

Triple-S XML version 2.0.001 (December 2006), pp 42 ff.
Household #2 as 5 LR Trees

Household 2
Semi-Det, South

Person 1
Male 1
>65 4
Work 2
CarD 1

Person 2
Fem 2
46-65 3
Work 2
CarD 1

Person 3
Male 1
<21 1
Soc 1
CarP 2

a: Person: a1a2a3

b: Gender: ab1ab2ab1

b: Age: ab4ab3ab1

b: Purpose: ab2b2b1aab1

c: Mode: abc1bc1bc1aabc2

One tree per level requires 3 parallel b levels
Household #2 as 3 LR Trees

- Store upper level data instead of just the nodes.
- 3 parallel b levels, so need at least 3 trees

Gender: a1b1a2b2a3b1
Age: a1b4a2b3a3b1
Trips: a1b2c1b2c1b1c1a2a3b1c2
Tree vs Hierarchy of Surveys

• The three parallel levels mandate three storage instances for both – either three trees, or three survey files

• Left-right trees need to duplicate the upper paths for parallel levels

• But for circumstances where there are no parallel levels, such as Brand/Attribute/Ratings or Brand Image, left-right trees offer several advantages.

• The primary advantage is dramatically reduced storage requirements for typical brand-oriented consumer surveys
Left-right trees can also be used to store grids, cubes, or any N-dimensional data structure.

BrandX rated 5
BrandY rated 3
BrandZ rated 7
Multi-response

Brand Image  a1b1;2;3;4;5;6;7;8a2b5;6;7;8a3b2;3;5;6

1 BrandX
   1 Well-Known
   2 Trustworthy
   3 Easy to get
   4 High quality
   5 Good reputation
   6 Used by Professionals
   7 Modern
   8 Value for Money

2 BrandY
   5 Good reputation
   6 Used by Professionals
   7 Modern
   8 Value for Money

3 BrandZ
   2 Trustworthy
   3 Easy to get
   5 Good reputation
   6 Used by Professionals

• Note the ; delimiter to avoid confusion with European , as decimal place

• Any level (or dimension) can be multi-response, eg a1;2b3;4c5;6;7

• For 10 statements coded 1 to 10, the flat storage for 3 brands (spread format) requires 60 columns

• Can have multi-response at any level, eg a1;2b3;4;5
Current Grid/Cube Storage

The implementer must choose between

- traditional flat storage, or
- SSS ver 2.0 hierarchic storage

But a typical brand tracker will have many grids, cubes, etc – a random sample of 3 jobs gives, 15, 42, and 37 instances. The cost is either

- A large number of columns (if flat), or
- A large number of files (if SSS hierarchic)

And with internet collection now dominant, the tendency to allow responses for any subset of brands for which there is awareness (rather than just the traditional main brand list) can result in combinatorial explosions which impose a heavy burden on storage, RAM and CPU. International jobs also can have very large brand lists.

Real-world examples follow:
FMCG (1): Hierarchy of Surveys

SSS fixed-width export from Confirmit, 180 respondents, 12 brands, 10 grids and 5 cubes requires $15 \times 2 = 30$ files (15 XML, 15 ASC)

Comparing storage requirements:

<table>
<thead>
<tr>
<th>ASC</th>
<th>Bytes</th>
<th>Tree</th>
<th>Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data_0</td>
<td>15,747</td>
<td>B32</td>
<td>15,755</td>
</tr>
<tr>
<td>Data_1</td>
<td>14,728</td>
<td>B41</td>
<td>1,181</td>
</tr>
<tr>
<td>Data_2</td>
<td>38,523</td>
<td>B42a</td>
<td>492</td>
</tr>
<tr>
<td>Data_3</td>
<td>12,549</td>
<td>KC32</td>
<td>11,333</td>
</tr>
<tr>
<td>Data_4</td>
<td>9,218</td>
<td>KC41</td>
<td>862</td>
</tr>
<tr>
<td>Data_5</td>
<td>55,215</td>
<td>KC42a</td>
<td>537</td>
</tr>
<tr>
<td>Data_6</td>
<td>17,031</td>
<td>M32</td>
<td>14,469</td>
</tr>
<tr>
<td>Data_7</td>
<td>11,308</td>
<td>M41</td>
<td>975</td>
</tr>
<tr>
<td>Data_8</td>
<td>86,031</td>
<td>M42a</td>
<td>657</td>
</tr>
<tr>
<td>Data_9</td>
<td>18,321</td>
<td>P32</td>
<td>17,417</td>
</tr>
<tr>
<td>Data_10</td>
<td>18,528</td>
<td>P41</td>
<td>1,349</td>
</tr>
<tr>
<td>Data_11</td>
<td>68,055</td>
<td>P42a</td>
<td>594</td>
</tr>
<tr>
<td>Data_12</td>
<td>11,325</td>
<td>SP32</td>
<td>12,448</td>
</tr>
<tr>
<td>Data_13</td>
<td>9,978</td>
<td>SP41</td>
<td>968</td>
</tr>
<tr>
<td>Data_14</td>
<td>32,103</td>
<td>SP42a</td>
<td>465</td>
</tr>
<tr>
<td>total</td>
<td>418,660</td>
<td></td>
<td>79,502</td>
</tr>
</tbody>
</table>

A small number of brands, and high instantiation, but still five times less space
323 brands by 58 statements (multi-response) over 69,841 cases

- Spread format:
  Requires 323*58*2 = 37,468 columns
  columns * cases = 2,496 meg

- Bit format (divide by 2):
  Requires 323*58 = 18,734 columns
  columns * cases = 1,248 meg

- Tree as Fixed Width:
  Longest response = 1150 characters
  chars * cases = 76.6 meg

- Tree as Delimited:
  Sum of response lengths = 11.33 meg
FMCG (3) Fixed Width: Brand Statement Rating

204 brands by 4 statements by 5 ratings over 1,530 cases

- **Bit format:**
  Requires $204 \times 4 \times 5 = 4,080$ columns
  columns * cases = 6,096 k

- **Spread format:**
  Requires $204 \times 4 = 816$ columns
  columns * cases = 1,219 k

- **Tree as Fixed Width:**
  Longest response = 120 characters
  chars * cases = 179.3 k

- **Tree as Delimited:**
  Sum of response lengths = 51.5 k
Brand Rating:

```xml
<tree ident="BRAT">
  <position start="3" finish="10"/>
  <level ident="Brand" type="single">
    <values>
      <value code="1">AMEX</value>
      <value code="2">Visa</value>
    </values>
  </level>
  <level ident="Rating" type="single">
    <values>
      <value code="1">1</value>
      <value code="2">2</value>
      <value code="3">3</value>
    </values>
  </level>
</tree>
```

- New tag type, tree
- Different context for the `<level>` tag
- No href or parent, so the levels are subordinate

Column: 12345678901
Case#1: xxxab3a2b1x
Case#2: xxxa2b2 x
Case#3: xx x
Case#4: xxxa1b1a2b3x
Proposed SSS Storage: Delimited Single

Brand Rating:
<tree ident="BRAT">
  <position start="3"/>
  <level ident="Brand" type="single">
    <values>
      <value code="1">AMEX</value>
      <value code="2">Visa</value>
    </values>
  </level>
  <level ident="Rating" type="single">
    <values>
      <value code="1">1</value>
      <value code="2">2</value>
      <value code="3">3</value>
    </values>
  </level>
</variable>

Column: 12345678901234
Case#1: x,x,a1b3a2b1,x
Case#2: x,x,a2b2,x
Case#3: x,x,,x
Case#4: x,x,a1b1a2b3,x
Proposed SSS Storage: Fixed Width Multi

Brand Image:

```xml
<tree ident="BIM">
    <position start="3" finish="12"/>
    <level ident="Brand" type="single">
        <values>
            <value code="1">AMEX</value>
            <value code="2">Visa</value>
        </values>
    </level>
    <level ident="Image" type="multiple">
        <values>
            <value code="1">Cool</value>
            <value code="2">Relevant</value>
            <value code="3">Popular</value>
        </values>
    </level>
</tree>
```

<table>
<thead>
<tr>
<th>Column</th>
<th>Case#1</th>
<th>Case#2</th>
<th>Case#3</th>
<th>Case#4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>xxxa1b1;3a2b1</td>
<td>xxxa2b1;2;3</td>
<td>xx</td>
<td>xxxa1b1a2b1;2x</td>
</tr>
</tbody>
</table>
Proposed SSS Storage: Delimited Multi

Brand Image:

```xml
<tree ident="BIM">
  <position start="3"/>
  <level ident="Brand" type="single">
    <values>
      <value code="1">AMEX</value>
      <value code="2">Visa</value>
    </values>
    <level ident="Image" type="multiple">
      <values>
        <value code="1">Cool</value>
        <value code="2">Relevant</value>
        <value code="3">Popular</value>
      </values>
    </level>
  </level>
</tree>
```

<table>
<thead>
<tr>
<th>Column:</th>
<th>1234567890123456</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case#1</td>
<td>x,x,a1b1;3a2b1,x</td>
</tr>
<tr>
<td>Case#2</td>
<td>x,x,a2b1;2;3,x</td>
</tr>
<tr>
<td>Case#3</td>
<td>x,x,,x</td>
</tr>
<tr>
<td>Case#4</td>
<td>x,x,a1b1a2b1;2,x</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case#1</th>
<th>x,x,a1b1;3a2b1,x</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case#2</td>
<td>x,x,a2b1;2;3,x</td>
</tr>
<tr>
<td>Case#3</td>
<td>x,x,,x</td>
</tr>
<tr>
<td>Case#4</td>
<td>x,x,a1b1a2b1;2,x</td>
</tr>
</tbody>
</table>
Pros and Cons

Pro:
• 2 files only always (one XML, one ASC)
• No need for Link IDs
• No need for <parent> and <href> tags
• No need for a Definition XML
• The number of cases (across all data) remains constant
• The base counts at each level are simply the number of a-nodes, b-nodes, c-nodes etc
• Data is implicitly ordered, so do not need order attribute
• Dramatic reduction in space requirements for grids/cubes with large codeframes when only a subset have responses, especially under CSV
• The current <Hierarchy> tags are unaffected
• A levels job can store grids and cubes asked at different levels as trees, avoiding levels within levels conundrums

Con:
• Could cost more characters than fixed-width for node-complete (all codes at all levels are instantiated)
• Position is recorded only for the start/end of the tree
Household Data Storage

XML could look like this:

```xml
<trees ident="HHTrips">
  <level ident="Person" type="single">
    <position start="1" />
    <values>
      <range from="1" to="10" />
    </values>
  </level>
  <level ident="Gender" type="single" parent="Person">
    <position start="2" />
    <values>
      <value code="1">Male</value>
      <value code="2">Female</value>
    </values>
  </level>
  <level ident="Age" type="single" parent="Person">
    <position start="3" />
    <values>
      <value code="1">Under 21</value>
      <value code="2">21-45</value>
      <value code="3">46-65</value>
      <value code="4">Over 65</value>
    </values>
  </level>
  <level ident="Purpose" type="single" parent="Person">
    <position start="4" />
    <values>
      <value code="1">Social</value>
      <value code="2">Work</value>
      <value code="3">Business</value>
    </values>
  </level>
  <level ident="Method" type="single" parent="Purpose">
    <position start="5" />
    <values>
      <value code="1">Car Driver</value>
      <value code="2">Car Passenger</value>
      <value code="3">Bus</value>
      <value code="4">Train</value>
    </values>
  </level>
</trees>
```

HH#1: a1a2,ab2ab1,ab2ab2,ab1b1,abc3bc2
HH#2: a1a2a3, ab1ab2ab1, ab4ab3ab1, ab2b2b1aab1, abc1bc1bc1aabc2
HH#3: a1,ab2,ab2,ab2b2,abc3bc3

• Trees tag because a set of trees is described
• The top level Person has no parent
• The parent attribute allows parallelism
• If no parents assigned then same as <tree>
End