The Data Cube as a Typed Linear Algebra Operator

DBPL 2017 — 16th Symp. on DB Prog. Lang.

Technische Universität München (TUM), 1st Sep 2017

(Also: 38th InfoBlender Seminar, HASLab, 27th Sep 2017)

J.N. Oliveira

HASLab
INESC TEC & U.Minho
(H2020-732051: CloudDBAppliance)

H.D. Macedo

SW Eng Group @ U.Aharus
The 16th International Symposium on Database Programming Languages (DBPL 2017) will be held in conjunction with VLDB 2017 on September 1st, 2017, in Munich, Germany.
“Only by taking infinitesimally small units for observation (the differential of history, that is, the individual tendencies of men) and attaining to the art of integrating them (that is, finding the sum of these infinitesimals) can we hope to arrive at the laws of history.”

Leo Tolstoy, “War and Peace”
- Book XI, Chap.II (1869)

150 years later, this is what we are trying to attain through data-mining.

But — how fit are our maths for the task?

Have we attained the “art of integration”?

L. Tolstoy (1828–1910)
Since the early days of psychometrics in the social sciences (1970s), linear algebra (LA) has been central to data analysis (e.g. tensor decompositions etc)

We follow this trend but in a typed way, merging LA with polymorphic type systems, over a categorial basis.

We address a concrete example: that of studying the maths behind a well-known device in data analysis, the data cube construction.

We will define this construction as a polymorphic LA operator.

Typing linear algebra is proposed as a strategy for achieving such an “art of integration”.

Running example

Raw data:

<table>
<thead>
<tr>
<th>#</th>
<th>Model</th>
<th>Year</th>
<th>Color</th>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chevy</td>
<td>1990</td>
<td>Red</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Chevy</td>
<td>1990</td>
<td>Blue</td>
<td>87</td>
</tr>
<tr>
<td>3</td>
<td>Ford</td>
<td>1990</td>
<td>Green</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>Ford</td>
<td>1990</td>
<td>Blue</td>
<td>99</td>
</tr>
<tr>
<td>5</td>
<td>Ford</td>
<td>1991</td>
<td>Red</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Ford</td>
<td>1991</td>
<td>Blue</td>
<td>7</td>
</tr>
</tbody>
</table>

$t = (\text{Raw data})$

**Rows** — records ($n$-many) — the *infinitesimals*

**Columns** — attributes — the *observables*

**Column-orientation** — each column (attribute) $A$ represented by a function $t_A : n \to A$ such that $a = t_A(i)$ means “$a$ is the value of attribute $A$ in record nr $i$”.
Can records be rebuilt from such attribute projection functions?

Yes — by **tupling** them.

---

**Tupling**: Given functions \( f : A \rightarrow B \) and \( g : A \rightarrow C \), their tupling is the function \( f \uplus g \) such that

\[
(f \uplus g) a = (f a, g a)
\]

For instance,

\[
(t_{\text{Color}} \uplus t_{\text{Model}}) 2 = (\text{Blue}, \text{Chevy}),
(t_{\text{Year}} \uplus (t_{\text{Color}} \uplus t_{\text{Model}})) 3 = (1990, (\text{Green}, \text{Ford}))
\]

and so on.
For the column-oriented model to work one will need to express \textit{joins}, and these call for \textit{“inverse”} functions, e.g.

\[(t_{Model} \sqcap t_{Year})^\circ (Ford, 1990) = \{3, 4\}\]

meaning that tuples nr 3 and nr 4 have the same model \textit{(Ford)} and year \textit{(1990)}.

However, the type \(f^\circ : A \rightarrow \mathcal{P} n\) is rather annoying, as it involves \textbf{sets} of tuple indices — these will add an extra layer of complexity.

Fortunately, there is a simpler way — \textbf{typed linear algebra}, also known as \textbf{linear algebra of programming (LAoP)}. 
The LAoP approach

Represent functions by Boolean matrices:

Given (finite) types $A$ and $B$, any function $f : A \rightarrow B$
can be represented by a matrix $[f]$ with $A$-many columns
and $B$-many rows such that, for any $b \in B$ and $a \in A$,
the $(b, a)$-matrix-cell is

$$b \ [f] \ a = \begin{cases} 1 & \iff b = f \ a \\ 0 & \text{otherwise} \end{cases}$$

**NB**: Following the infix notation usually adopted for relations (which are
Boolean matrices) — for instance $y \leq x$ — we write $y \ M \ x$ to denote
the contents of the cell in matrix $M$ addressed by row $y$ and column $x$. 
The LAoP approach

One projection function (matrix) per **dimension** attribute:

<table>
<thead>
<tr>
<th>$t_{Model}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevy</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ford</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t_{Year}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1990</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1991</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$t_{Color}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Green</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Red</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Model</th>
<th>Year</th>
<th>Color</th>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chevy</td>
<td>1990</td>
<td>Red</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Chevy</td>
<td>1990</td>
<td>Blue</td>
<td>87</td>
</tr>
<tr>
<td>3</td>
<td>Ford</td>
<td>1990</td>
<td>Green</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>Ford</td>
<td>1990</td>
<td>Blue</td>
<td>99</td>
</tr>
<tr>
<td>5</td>
<td>Ford</td>
<td>1991</td>
<td>Red</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Ford</td>
<td>1991</td>
<td>Blue</td>
<td>7</td>
</tr>
</tbody>
</table>

**NB:** we tend to abbreviate $[f]$ by $f$ when the context is clear.
The LAoP approach

Note how the inverse of a function is also represented by a Boolean matrix, e.g.

\[
\begin{array}{c|cc}
 t^o_{Model} & \text{Chevy} & \text{Ford} \\
1 & 1 & 0 \\
2 & 1 & 0 \\
3 & 0 & 1 \\
4 & 0 & 1 \\
5 & 0 & 1 \\
6 & 0 & 1 \\
\end{array}
\]

versus

\[
\begin{array}{cccccccc}
 t_{Model} & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{Chevy} & 1 & 1 & 0 & 0 & 0 & 0 \\
\text{Ford} & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

— no need for powersets.

Clearly,

\[
j t^o_{Model} a = a t_{Model} j
\]

Given a matrix \( M \), \( M^o \) is known as the transposition of \( M \).
The LAoP approach

We **type** matrices in the same way as functions: \( M : A \rightarrow B \) means a matrix \( M \) with \( A \)-many columns and \( B \)-many rows.

Matrices are arrows: \( A \xrightarrow{M} B \) denotes a matrix from \( A \) (source) to \( B \) (target), where \( A, B \) are (finite) types.

Writing \( B \leftarrow M \ A \) means the same as \( A \xrightarrow{M} B \).

**Composition — aka matrix multiplication:**

\[
\begin{align*}
  B & \leftarrow M \ A \leftarrow N \ C \\
  b(M \cdot N)c & = \langle \sum a :: (b M a) \times (a N c) \rangle
\end{align*}
\]
The LAoP approach

**Function composition** implemented by matrix multiplication,

\[
[f \cdot g] = [f] \cdot [g]
\]

**Identity** — the identity matrix \(id\) corresponds to the identity function and is such that

\[
M \cdot id = M = id \cdot M
\]  

(1)

**Function tupling** corresponds to the so-called **Khatri-Rao product** \(M \triangleleft N\) defined index-wise by

\[
(b, c) (M \triangleleft N) a = (b M a) \times (c N a)
\]

(2)

Khatri-Rao is a “column-wise” version of the well-known **Kronecker product** \(M \otimes N\):

\[
(y, x) (M \otimes N) (b, a) = (y M b) \times (x N a)
\]

(3)
Typing data

The raw data given above is represented in the LAoP by the expression

\[ \mathbf{v} = (t_{\text{Year}} \, \triangledown \, (t_{\text{Color}} \, \triangledown \, t_{\text{Model}})) \cdot (t^{\text{Sale}})^{\circ} \]

of type

\[ \mathbf{v} : 1 \rightarrow (\text{Year} \times (\text{Color} \times \text{Model})) \]

depicted aside.

\[ \mathbf{v} \] is a **multi-dimensional** column vector — a **tensor**. Datatype \( 1 = \{\text{ALL}\} \) is the so-called **singleton** type.
**Dimensions and measures**

*Sale* is a special kind of data — a measure. Measures are encoded as row vectors, e.g.

\[
\begin{array}{cccccc}
\text{Model} & \text{Year} & \#t & \text{Color} & t_{\text{Sale}} \\
1 & 1990 & 5 & 67 & 64 & 99 & 8 & 7 \\
\end{array}
\]

Recall

<table>
<thead>
<tr>
<th>#</th>
<th>Model</th>
<th>Year</th>
<th>Color</th>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Chevy</td>
<td>1990</td>
<td>Red</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>Chevy</td>
<td>1990</td>
<td>Blue</td>
<td>87</td>
</tr>
<tr>
<td>3</td>
<td>Ford</td>
<td>1990</td>
<td>Green</td>
<td>64</td>
</tr>
<tr>
<td>4</td>
<td>Ford</td>
<td>1990</td>
<td>Blue</td>
<td>99</td>
</tr>
<tr>
<td>5</td>
<td>Ford</td>
<td>1991</td>
<td>Red</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Ford</td>
<td>1991</td>
<td>Blue</td>
<td>7</td>
</tr>
</tbody>
</table>

Summary:
- dimensions are matrices, measures are vectors.
- Measures provide for integration in Tolstoy’s sense — aka consolidation.
Totalisers

There is a unique function in type $A \rightarrow 1$, usually named $A \overset{!}{\rightarrow} 1$. This corresponds to a row vector wholly filled with 1s.

Example: $2 \overset{!}{ightarrow} 1 = [1 \quad 1]

Given $M : B \rightarrow A$, the expression $! \cdot M$ (where $A \overset{!}{\rightarrow} 1$) is the row vector (of type $B \rightarrow 1$) that contains all column totals of $M$,

$[1 \quad 1] \cdot \begin{bmatrix} 50 & 40 & 85 & 115 \\ 50 & 10 & 85 & 75 \end{bmatrix} = [100 \quad 50 \quad 170 \quad 190]

Given type $A$, define its totalizer matrix $A \overset{\tau_A}{\rightarrow} A + 1$ by

$\tau_A : A \rightarrow A + 1$

$\tau_A = \begin{bmatrix} id \\ ! \end{bmatrix}$

(5)

Thus $\tau_A \cdot M$ yields a copy of $M$ on top of the corresponding totals.
Cubes

Data **cubes** are easily obtained from products of totalizers.

Recall the Kronecker (tensor) product $M \otimes N$ of two matrices:

\[
\begin{array}{ccc}
A & C & A \times C \\
M & N & M \otimes N \\
B & D & B \times D \\
\end{array}
\]

The matrix

\[
A \times B \xrightarrow{\tau_A \otimes \tau_B} (A + 1) \times (B + 1)
\]

provides for totalization on the **two dimensions** $A$ and $B$.

Indeed, type $(A + 1) \times (B + 1)$ is isomorphic to $A \times B + A + B + 1$, whose four parcels represent the four elements of the “**dimension powerset** of \{A, B\}”.
Cube = muti-dimensional totalisation

Recalling

\[ \mathbf{v} = (t_{Year} \times (t_{Color} \times t_{Model})) \cdot (t^{Sale})^\circ \]

we build

\[ \mathbf{c} = (\tau_{Year} \otimes (\tau_{Color} \otimes \tau_{Model})) \cdot \mathbf{v} \]

This is the multidimensional vector (tensor) representing the data cube for

- **dimensions** \( Year, Color, Model \)
- **measure** \( Sale \)

depicted aside.
Totalisers yield cubes

Thanks to \textit{\( \times \)-absorption}

\[ (M \otimes N) \cdot (P \downarrow Q) = (M \cdot P) \downarrow (N \cdot Q) \]  

we can simplify the definition:

\[ c \quad = \quad (\tau_{\text{Year}} \otimes (\tau_{\text{Color}} \otimes \tau_{\text{Model}})) \cdot v \]

\[ = \quad \{ \quad v = (t_{\text{Year}} \downarrow (t_{\text{Color}} \downarrow t_{\text{Model}})) \cdot (t_{\text{Sale}}) \circ \quad \} \]

\[ (\tau_{\text{Year}} \otimes (\tau_{\text{Color}} \otimes \tau_{\text{Model}})) \cdot (t_{\text{Year}} \downarrow (t_{\text{Color}} \downarrow t_{\text{Model}})) \cdot (t_{\text{Sale}}) \circ \]

\[ = \quad \{ \quad \text{absorption-law (6)} \quad \} \]

\[ ((\tau_{\text{Year}} \cdot t_{\text{Year}}) \downarrow ((\tau_{\text{Color}} \cdot t_{\text{Color}}) \downarrow ((\tau_{\text{Model}} \cdot t_{\text{Model}})))) \cdot (t_{\text{Sale}}) \circ \]

\[ = \quad \{ \quad \text{define } t'_A = \tau_A \cdot t_A \quad \} \]

\[ (t'_{\text{Year}} \downarrow (t'_{\text{Color}} \downarrow t'_{\text{Model}})) \cdot (t_{\text{Sale}}) \circ \]

Note that \( t'_A = \left[ \frac{t_A}{1} \right] \), since \( t_A \) is a function.
In our approach a **cube** is not necessarily one such column vector.

The key to **generic** data cubes is (generalized) **vectorization**, a kind of "**matrix currying**": given \( A \times B \xrightarrow{M} C \) with \( A \times B \)-many columns and \( C \)-many rows, reshape \( M \) into its **vectorized** version \( B \xrightarrow{\text{vec}_A M} A \times C \) with \( B \)-many columns and \( A \times C \)-many rows.

Such matrices, \( M \) and \( \text{vec}_A M \), are **isomorphic** in the sense that they contain the same information in different formats, cf

\[
c \cdot M (a, b) = (a, c) \cdot (\text{vec}_A M) \cdot b
\]

which holds for every \( a, b, c \).
Generalizing data cubes

**Vectorization** thus has an inverse operation — **unvectorization**:

\[ A \times B \rightarrow C \quad \cong \quad B \rightarrow A \times C \]

That is, \( M \) can be retrieved back from \( \text{vec}_A M \) by unvectorizing it:

\[ N = \text{vec}_A M \iff \text{unvec}_A N = M \]  \hspace{1cm} (8)

Vectorization has a rich algebra, e.g. a **fusion**-law

\[ (\text{vec} M) \cdot N = \text{vec} (M \cdot (id \otimes N)) \]  \hspace{1cm} (9)

and an **absorption**-law:

\[ \text{vec} (M \cdot N) = (id \otimes M) \cdot \text{vec} N \]  \hspace{1cm} (10)
Let us unvectorize our starting (data) tensor, across dimension \textit{Year}:

\[
\text{unvec}_{\text{Year}}(\text{Year} \times (\text{Color} \times \text{Model})) \leftarrow 1
\]

\[
\begin{pmatrix}
  \text{ALL} \\
  \text{Blue} \\
  \text{Green} \\
  \text{Red}
\end{pmatrix}
\begin{pmatrix}
  \text{Chevy} & 87 \\
  \text{Ford} & 99
\end{pmatrix}
\begin{pmatrix}
  \text{Chevy} & 0 \\
  \text{Ford} & 64
\end{pmatrix}
\begin{pmatrix}
  \text{Chevy} & 5 \\
  \text{Ford} & 0
\end{pmatrix}
\begin{pmatrix}
  \text{Chevy} & 0 \\
  \text{Ford} & 7
\end{pmatrix}
\begin{pmatrix}
  \text{Chevy} & 0 \\
  \text{Ford} & 0
\end{pmatrix}
\begin{pmatrix}
  \text{Chevy} & 0 \\
  \text{Ford} & 8
\end{pmatrix}
\]

\[
\begin{pmatrix}
  1990 & 1991 \\
  \text{Blue} & \text{Chevy} & 87 & 0 \\
  \text{Ford} & 99 & 7
\end{pmatrix}
\begin{pmatrix}
  \text{Green} & \text{Chevy} & 0 & 0 \\
  \text{Ford} & 64 & 0
\end{pmatrix}
\begin{pmatrix}
  \text{Red} & \text{Chevy} & 5 & 0 \\
  \text{Ford} & 0 & 8
\end{pmatrix}
\]

There is room for further unvectorizing the outcome, this time across \textit{Color} — next slide:
(De)vectorization

Further unvectorization:

\[
\begin{pmatrix}
\text{Color} \times \text{Model} & \text{Year} \\
\text{Blue} & \text{Chevy} & 87 & 0 \\
& \text{Ford} & 99 & 7 \\
\text{Green} & \text{Chevy} & 0 & 0 \\
& \text{Ford} & 64 & 0 \\
\text{Red} & \text{Chevy} & 5 & 0 \\
& \text{Ford} & 0 & 8
\end{pmatrix}
\]

\[
\begin{array}{cccc}
\text{Model} \leftrightarrow \text{Color} \times \text{Year} \\
\text{Blue} & \text{Green} & \text{Red} \\
\text{Chevy} & 87 & 0 & 0 & 0 & 5 & 0 \\
\text{Ford} & 99 & 7 & 64 & 0 & 0 & 8
\end{array}
\]

and so on.
Generic cubes

It turns out that cubes can be calculated for any such two-dimensional versions of our original data tensor, for instance,

\[
\text{cube } N : \text{Model} + 1 \leftarrow (\text{Color} + 1) \times (\text{Year} + 1)
\]

\[
\text{cube } N = \tau_{\text{Model}} \cdot N \cdot (\tau_{\text{Color}} \otimes \tau_{\text{Year}})^\circ
\]

where \( N \) stands for the second matrix of the previous slide, yielding

<table>
<thead>
<tr>
<th></th>
<th>Blue</th>
<th></th>
<th>Green</th>
<th></th>
<th>Red</th>
<th></th>
<th>ALL</th>
<th></th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevy</td>
<td>87</td>
<td>0</td>
<td>87</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Ford</td>
<td>99</td>
<td>7</td>
<td>106</td>
<td>64</td>
<td>0</td>
<td>64</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>ALL</td>
<td>186</td>
<td>7</td>
<td>193</td>
<td>64</td>
<td>0</td>
<td>64</td>
<td>5</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>92</td>
<td>0</td>
<td>92</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
<td>92</td>
</tr>
<tr>
<td></td>
<td>163</td>
<td>15</td>
<td>178</td>
<td>15</td>
<td>178</td>
<td>15</td>
<td>255</td>
<td>15</td>
<td>270</td>
</tr>
</tbody>
</table>

The 36 entries of the original cube have been rearranged in a 3*12 rectangular layout, as dictated by the dimension cardinalities.
The **cube** (LA) operator

### Definition (Cube)

Let $M$ be a matrix of type

$$\prod_{j=1}^{n} B_j \xleftarrow{M} \prod_{i=1}^{m} A_i \quad (11)$$

We define matrix **cube** $M$, the *cube of* $M$, as follows

$$\text{cube } M = \left( \bigotimes_{j=1}^{n} \tau_{B_j} \right) \cdot M \cdot \left( \bigotimes_{i=1}^{m} \tau_{A_i} \right)^\circ \quad (12)$$

where $\bigotimes$ is finite Kronecker product.

So **cube** $M$ has type $\prod_{j=1}^{n} (B_j + 1) \xleftarrow{M} \prod_{i=1}^{m} (A_i + 1)$.

□
Properties of data cubing

Linearity:

\[ \text{cube} \ (M + N) = \text{cube} \ M + \text{cube} \ N \]  

(13)

Proof: Immediate by bilinearity of matrix composition:

\[ M \cdot (N + P) = M \cdot N + M \cdot P \]  

(14)

\[ (N + P) \cdot M = N \cdot M + P \cdot M \]  

(15)

This can be taken advantage of not only in incremental data cube construction but also in parallelizing data cube generation.
Properties of data cubing

Updatability: by Khatri-Rao product linearity,

\[(M + N) \triangleleft P = M \triangleleft P + N \triangleleft P\]

\[P \triangleleft (M + N) = P \triangleleft M + P \triangleleft N\]

the cube operator commutes with the usual CRUDE operations, namely with record updating. For instance, suppose record

<table>
<thead>
<tr>
<th>#</th>
<th>Model</th>
<th>Year</th>
<th>Color</th>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Ford</td>
<td>1991</td>
<td>Red</td>
<td>8</td>
</tr>
</tbody>
</table>

is updated to

<table>
<thead>
<tr>
<th>#</th>
<th>Model</th>
<th>Year</th>
<th>Color</th>
<th>Sale</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Chevy</td>
<td>1991</td>
<td>Red</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(t_{Model})</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevy</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ford</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(t'_{Model})</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevy</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Ford</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
One just has to compute the “delta” projection,

$$\delta_{Model} = t'_{Model} - t_{Model} =$$

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chevy</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Ford</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>

then the “delta cube”,

$$d = (\tau_{Year} \otimes (\tau_{Color} \otimes \tau_{Model})) \cdot v'$$

where

$$v' = (t_{Year} \downarrow (t_{Color} \downarrow \delta_{Model})) \cdot (t^{Sale})^o$$

and finally add the “delta cube” to the original cube:

$$c' = c + d.$$
Properties of data cubing

Theorem (Cube commutes with vectorization)

Let $X \leftarrow^M Y \times C$ and $Y \times X \leftarrow^{\text{vec } M} C$ be its $Y$-vectorization. Then

$$\text{vec}(\text{cube } M) = \text{cube}(\text{vec } M)$$

(16)

holds.

\[\square\]

The proof (in the paper) relies on the type diagrams:

\[
\begin{align*}
Y \times X & \leftarrow^{\text{vec}_Y M} C & \cong & & X \leftarrow^M Y \times C \\
(Y + 1) \times (X + 1) \lessapprox_{\text{vec}_{Y+1} (\text{cube } M)} C + 1 & \cong & & (Y + 1) \times (C + 1) \leftarrow^{\text{cube } M}
\end{align*}
\]
Properties of data cubing

The following theorem shows that changing the dimensions of a data cube does not change its totals.

**Theorem (Free theorem)**

Let \( B \leftarrow^M A \) be cubed into \( B + 1 \leftarrow^{\text{cube } M} A + 1 \), and \( r : C \to A \) and \( s : D \to B \) be arbitrary functions. Then

\[
\text{cube} \ (s \circ M \circ r) = (s \circ \text{id}) \circ (\text{cube } M) \circ (r \circ \text{id})
\]

(17)

holds, where \( M \oplus N = \begin{bmatrix} M & 0 \\ 0 & N \end{bmatrix} \) is matrix direct sum.

\( \square \)

The proof given in the paper resorts to the free theorem of polymorphic operators popularized by Wadler (1989) under the heading *Theorems for free!*.
Cube universality — slicing

**Slicing** is a specialized filter for a particular value in a dimension.

Suppose that from our starting cube

\[ c : 1 \to (Year + 1) \times ((Color + 1) \times (Model + 1)) \]

one is only interested in the data concerning year **1991**.

It suffices to regard data values as (categorial) **points**: given \( p \in A \), constant function \( p : 1 \to A \) is said to be a point of \( A \), for instance

\[
1991 : 1 \to Year + 1
\]

\[
1991 = \begin{bmatrix}
0 \\
1 \\
0
\end{bmatrix}
\]
Cube universality — slicing

Example:

\[ 1 \rightarrow c \rightarrow (\text{Year} + 1) \times ((\text{Color} + 1) \times (\text{Model} + 1)) \]

\[ 1 \times ((\text{Color} + 1) \times (\text{Model} + 1)) \]

\[ 1991^\circ \otimes \text{id} \]

\[ = \begin{bmatrix} 0 \\ 7 \\ 7 \\ 0 \\ 0 \\ 0 \\ 8 \\ 8 \\ 0 \\ 15 \\ 15 \end{bmatrix} \]
Gray et al. (1997) say that *going up the levels [of aggregated data] is called rolling-up*. 

In this sense, a **roll-up** operation over dimensions $A$, $B$ and $C$ could be the following form of (increasing) summarization:

\[
A \times (B \times C) \\
A \times B \\
A \\
1
\]

How does this work over a data cube? We take the simpler case of two dimensions $A$, $B$ as example.
Cube universality — rolling-up

The dimension powerset for $A$, $B$ is captured by the corresponding matrix injections onto the cube target type $(A + 1) \times (B + 1)$:

\[
\begin{align*}
(A + 1) \times (B + 1) \\
\theta & \quad \alpha \\
A \times B & \quad A & \quad B & \quad 1 \\
\beta & \quad \omega
\end{align*}
\]

where

\[
\begin{align*}
\theta &= i_1 \otimes i_1 \\
\alpha &= i_1 \vee i_2 \cdot ! \\
\beta &= i_1 \cdot ! \vee i_2 \\
\omega &= i_2 \vee i_2
\end{align*}
\]

**NB:** the injections $i_1$ and $i_2$ are such that $[i_1|i_2] = id$, where $[M|N]$ denotes the horizontal gluing of two matrices.
Cube universality — rolling-up

One can build compound injections, for instance

\[ \rho : (A + 1) \times (B + 1) \leftarrow A \times B + (A + 1) \]
\[ \rho = [\theta|\alpha|\omega] \]

Then, for \( M : C \rightarrow A \times B \):

\[ \rho^\circ \cdot (\text{cube } M) = \left[ \frac{M}{\bar{\text{fst}} \cdot M} \right] \cdot \tau_C \]

extracts from cube \( M \) the corresponding roll-up.

The next slides give a concrete example.
Cube universality — rolling-up

Let $M$ be the (generalized) data cube

<table>
<thead>
<tr>
<th></th>
<th>1990</th>
<th>1991</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Chevy</strong></td>
<td>87</td>
<td>0</td>
<td>87</td>
</tr>
<tr>
<td><strong>Blue</strong></td>
<td>99</td>
<td>7</td>
<td>106</td>
</tr>
<tr>
<td><strong>Ford</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ALL</strong></td>
<td>186</td>
<td>7</td>
<td>193</td>
</tr>
<tr>
<td><strong>Chevy</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td><strong>Green</strong></td>
<td>64</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td><strong>Ford</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ALL</strong></td>
<td>64</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td><strong>Chevy</strong></td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td><strong>Red</strong></td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td><strong>Ford</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>ALL</strong></td>
<td>5</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td><strong>Chevy</strong></td>
<td>92</td>
<td>0</td>
<td>92</td>
</tr>
<tr>
<td><strong>ALL</strong></td>
<td>163</td>
<td>15</td>
<td>178</td>
</tr>
<tr>
<td><strong>Ford</strong></td>
<td>255</td>
<td>15</td>
<td>270</td>
</tr>
</tbody>
</table>
Cube universality — rolling-up

Building the injection matrix \( \rho = [\theta|\alpha|\omega] \) for types \( \text{Color} \times \text{Model} + \text{Color} + 1 \rightarrow (\text{Color} + 1) \times (\text{Model} + 1) \) we get the following matrix (already transposed):

\[
\begin{array}{cccccccccccc}
\text{Blue} & \text{Green} & \text{Red} & \text{ALL} \\
\hline
\text{Chevy} & \text{Ford} & \text{ALL} & \text{Chevy} & \text{Ford} & \text{ALL} & \text{Chevy} & \text{Ford} & \text{ALL} & \text{Chevy} & \text{Ford} & \text{ALL} \\
\text{Blue} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
& 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Green} & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
& 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Red} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
& 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{Blue} & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\text{Green} & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
\text{Red} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
\text{ALL} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]
Cube universality — rolling-up

Then

\[
\rho^\circ \cdot \text{cube } M =
\]

<table>
<thead>
<tr>
<th></th>
<th>1990</th>
<th>1991</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chevy</td>
<td>87</td>
<td>0</td>
<td>87</td>
</tr>
<tr>
<td>Ford</td>
<td>99</td>
<td>7</td>
<td>106</td>
</tr>
<tr>
<td>Green</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chevy</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Ford</td>
<td>64</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>Red</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Chevy</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Ford</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Blue</td>
<td>186</td>
<td>7</td>
<td>193</td>
</tr>
<tr>
<td>Green</td>
<td>64</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>Red</td>
<td>5</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>ALL</td>
<td>255</td>
<td>15</td>
<td>270</td>
</tr>
</tbody>
</table>

Note how a roll-up is a particular “subset” of a cube.

Matrix \( \rho^\circ \) performs the (quantitative) selection of such a subset.
Summing up

Data science seems to be ignoring the role of types and type parametricity in software — one of the most significant advances in CS.

Nice theory called parametric polymorphism (John Reynolds, CMU).

So nice that you can derive properties of your operations solely by looking at their types 😊

As Kurt Lewin (1890-1947) once wrote it: "There is nothing more practical than a good theory".

J.C. Reynolds (1935–2013)
Summing up

Abadir and Magnus (2005) stress on the need for a standardized notation for linear algebra in the field of econometrics and statistics.

This talk suggests such a notation should be polymorphically typed.

Since (Macedo and Oliveira, 2013) the author has invested in typing linear algebra in a way that makes it closer to modern typed languages.

This extends previous efforts on applying LA to OLAP (Macedo and Oliveira, 2015)

(Still not convinced? Peek the next slide.)
Annex

(For those who care mostly about efficiency)

Aside: Plot taken from a recent MSc report on TPC-H benchmarking LA approach to analytical querying (on-going work).
References


