The Data Cube as a Typed Linear Algebra Operator

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

Technische Universität München (TUM), 1st Sep 2017

J.N. Oliveira

H.D. Macedo

INESC TEC & U.Minho

(H2020-732051: CloudDBAppliance)

SW Eng Group @ U.Aharus
“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”? 
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.

Typed linear algebra is proposed as a rich setting for such an “art of integration” to be achieved.
### 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>

\[ t = \]

Columns — attributes — the observables  
Rows — records \((n\text{-many})\) — the infinitesimals

**Column-orientation** — each column (attribute) \(A\) represented by a function \(t_A : n \rightarrow A\) such that \(a = t_A(i)\) means “\(a\) is the value of attribute \(A\) in record nr \(i\)”.
Records are tuples

Can records be rebuilt from such **attribute projection** functions?

Yes — by **tupling** them.

---

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

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

---

For instance,

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

and so on.
Inverting tuples

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

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

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

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

Fortunately, there is a simpler way — typed linear algebra, also known as 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 $a \in A$, matrix cell

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

**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.

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

— no need for powersets.

Clearly,

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

Given a matrix \( M \), \( M^\circ \) is known as the \textit{transposition} of \( M \).
The LAoP approach

We type matrices in the same way as functions: $M : A \to B$ 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 $B \xleftarrow{M} A$ means the same as $A \xrightarrow{M} B$.

Composition — aka matrix multiplication:

$$b(M \cdot N)c = \langle \sum a :: (b M a) \times (a N c) \rangle$$
**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

$$M \cdot id = M = id \cdot M \quad (1)$$

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

$$ (b, c) (M \triangleleft N) a = (b M a) \times (c N a) \quad (2) $$

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

$$ (y, x) (M \otimes N) (b, a) = (y M b) \times (x N a) \quad (3) $$
Typing data

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

$$\mathbf{v} = (t_{\text{Year}} \lor (t_{\text{Color}} \lor t_{\text{Model}})) \cdot (t_{\text{Sale}})^o$$

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.

\[
\begin{array}{c|cccccc}
  \text{t}^{\text{Sale}} & 1 & 2 & 3 & 4 & 5 & 6 \\
  \hline
  1 & 5 & 87 & 64 & 99 & 8 & 7 \\
\end{array}
\]

recall

\[
\begin{array}{cccc}
  \# & \text{Model} & \text{Year} & \text{Color} \\
  1 & \text{Chevy} & 1990 & \text{Red} \\
  2 & \text{Chevy} & 1990 & \text{Blue} \\
  3 & \text{Ford} & 1990 & \text{Green} \\
  4 & \text{Ford} & 1990 & \text{Blue} \\
  5 & \text{Ford} & 1991 & \text{Red} \\
  6 & \text{Ford} & 1991 & \text{Blue} \\
\end{array}
\]

\[
\begin{array}{cccc}
  \#t & \text{t}^{\text{Model}} & \text{t}^{\text{Year}} & \text{t}^{\text{Color}} \\
  \hline
  1 & & & \\
\end{array}
\]

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 $A \xrightarrow{!} 1$. This corresponds to a row vector wholly filled with 1s.

Example: $2 \xrightarrow{!} 1 = [1 \ 1]$

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

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

Given type $A$, define its totalizer matrix $A \xrightarrow{\tau_A} A + 1$ 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.
Data **cubes** can be obtained from products of totalizers.

Recall the Kronecker (tensor) product $M \otimes N$ of two matrices $A \xrightarrow{M} B$ and $C \xrightarrow{N} D$, which is of type $A \times C \xrightarrow{M \otimes N} B \times D$.

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 $\{A, B\}$".
Recalling

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

build

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

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

- **dimensions** Year, Color, Model
- **measure** Sale
depicted aside.
Totalisers yield cubes

We reason:

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

\[ = \begin{cases} v = (t_{\text{Year}} \triangledown (t_{\text{Color}} \triangledown t_{\text{Model}})) \cdot (t^{\text{Sale}})^\circ \\ (\tau_{\text{Year}} \otimes (\tau_{\text{Color}} \otimes \tau_{\text{Model}})) \cdot (t_{\text{Year}} \triangledown (t_{\text{Color}} \triangledown t_{\text{Model}})) \cdot (t^{\text{Sale}})^\circ \end{cases} \]

\[ = \begin{cases} \text{property } (M \otimes N) \cdot (P \triangledown Q) = (M \cdot P) \triangledown (N \cdot Q) \end{cases} \]

\[ = \begin{cases} ((t_{\text{Year}} \cdot t_{\text{Year}}) \triangledown ((t_{\text{Color}} \cdot t_{\text{Color}}) \triangledown ((t_{\text{Model}} \cdot t_{\text{Model}})))) \cdot (t^{\text{Sale}})^\circ \\ \text{define } t'_A = \tau_A \cdot t_A \end{cases} \]

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

Note that \( t'_A = \left[ \frac{t_A}{1} \right] \), since \( t_A \) is a function.
Generalizing data cubes

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, as

$$c \ M \ (a, b) \ = \ (a, c) \ (\text{vec}_A M) \ b$$

holds for every $a, b, c$. 
Generalizing data cubes

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

\[
\begin{array}{ccc}
A \times B & \rightarrow & C \\
& \cong \quad & B \rightarrow A \times C \\
& \leftarrow & \text{vec}_A \\
& \leftarrow & \text{unvec}_A
\end{array}
\]

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

\[
N = \text{vec}_A M \iff \text{unvec}_A N = M \tag{7}
\]

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

\[
(\text{vec } M) \cdot N = \text{vec } (M \cdot (id \otimes N)) \tag{8}
\]

and an **absorption**-law:

\[
\text{vec } (M \cdot N) = (id \otimes M) \cdot \text{vec } N \tag{9}
\]
(De)vectorization

Devectorizing our starting tensor, across dimension $\text{Year}$:

\[
\begin{pmatrix}
\text{Year} & \text{(Color $\times$ Model)} & \text{ALL} \\
1990 & \text{Blue} & \text{Chevy} & 87 \\
& & \text{Ford} & 99 \\
1991 & \text{Green} & \text{Chevy} & 5 \\
& & \text{Ford} & 64 \\
& \text{Red} & \text{Chevy} & 0 \\
& & \text{Ford} & 0 \\
\end{pmatrix}
\]

\[\text{unvec}_{\text{Year}}\]

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

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

There is room for further devectorizing the outcome, this time across $\text{Color}$ — next slide:
Further devectorization:

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

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,

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

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

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>
<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>5</td>
<td>0</td>
<td>5</td>
<td>92</td>
<td>0</td>
<td>92</td>
</tr>
<tr>
<td>Ford</td>
<td>99</td>
<td>7</td>
<td>106</td>
<td>64</td>
<td>0</td>
<td>8</td>
<td>8</td>
<td>163</td>
<td>15</td>
<td>178</td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>186</td>
<td>7</td>
<td>193</td>
<td>64</td>
<td>0</td>
<td>8</td>
<td>13</td>
<td>255</td>
<td>15</td>
<td>270</td>
<td></td>
</tr>
</tbody>
</table>

See how 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

$$\prod_{j=1}^{n} B_j \leftarrow M \prod_{i=1}^{m} A_i$$  \hspace{1cm} (10)

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

$$\text{cube } M = \bigotimes_{j=1}^{n} \tau_{B_j} \cdot M \cdot \bigotimes_{i=1}^{m} \tau_{A_i}$$  \hspace{1cm} (11)

where $\bigotimes$ is finite Kronecker product.

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

Linearity:

\[ \text{cube} \ (M + N) = \text{cube} \ M + \text{cube} \ N \]  
\[ (12) \]

**Proof**: Immediate by bilinearity of matrix composition:

\[ M \cdot (N + P) = M \cdot N + M \cdot P \]  
\[ (13) \]
\[ (N + P) \cdot M = N \cdot M + P \cdot M \]  
\[ (14) \]

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

Updatability: by Khatri-Rao product linearity,

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

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

the **cube** operator commutes with the usual CRUDE operations, namely 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>
Properties of data cubing

One just has to compute the “delta” projection,

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

\[
\begin{array}{c|ccccccc}
\text{Chevy} & 1 & 2 & 3 & 4 & 5 & 6 \\
\text{Ford} & 0 & 0 & 0 & 0 & -1 & 0 \\
\end{array}
\]

then the “delta cube”,

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

where

$$v' = (t_{\text{Year}} \lozenge (t_{\text{Color}} \lozenge \delta_{\text{Model}})) \cdot (t_{\text{Sale}})^\circ$$

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

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

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

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

(15)

holds. □

Type diagrams:

\[
\begin{array}{ccc}
& & \\
& & \\
Y \times X & \leftarrow^{\text{vec}_Y M} & C \\
\tau_Y \otimes \tau_M & & \tau_C^o \\
& \uparrow & \\
(Y + 1) \times (X + 1) & \leftarrow_{\text{vec}_{Y+1}(\text{cube } M)} & C + 1 \\
& & \\
& & \\
& & \\
X & \leftarrow^M Y \times C \\
\tau_X & & (\tau_Y \otimes \tau_C)^o \\
& \downarrow & \\
X + 1 & \leftarrow^{\text{cube } M} (Y + 1) \times (C + 1) \\
\end{array}
\]

(Proof in the paper.)
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 \( r : C \rightarrow A \) and \( s : D \rightarrow B \) be arbitrary functions. Then

\[
\text{cube} (s \circ \cdot M \cdot r) = (s \circ \oplus \text{id}) \cdot (\text{cube} M) \cdot (r \circ \oplus \text{id}) \tag{16}
\]

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

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

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

one is only interested in the data concerning year 1991.

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

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

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

Example:

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

\[ 1991 \degree \otimes \text{id} \]

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

\[ = \]

\[
\begin{bmatrix}
0 \\
7 \\
7 \\
0 \\
0 \\
0 \\
8 \\
8 \\
0 \\
15 \\
15
\end{bmatrix}
\]
Cube universality — rolling-up

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 \textit{injections} onto the cube target type \((A + 1) \times (B + 1)\):

\[
\begin{array}{c}
(A + 1) \times (B + 1) \\
\theta \quad \alpha \quad \beta \quad \omega \\
A \times B \quad A \quad B \quad 1
\end{array}
\]

where

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

\textbf{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 $M : C \rightarrow A \times B$:

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

extracts from $\text{cube } M$ 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>Chevy</td>
<td>87</td>
<td>0</td>
<td>87</td>
</tr>
<tr>
<td>Blue</td>
<td>99</td>
<td>7</td>
<td>106</td>
</tr>
<tr>
<td>Ford</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>186</td>
<td>7</td>
<td>193</td>
</tr>
<tr>
<td>Chevy</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Green</td>
<td>64</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>Ford</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>64</td>
<td>0</td>
<td>64</td>
</tr>
<tr>
<td>Chevy</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>Red</td>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>Ford</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</td>
<td>5</td>
<td>8</td>
<td>13</td>
</tr>
<tr>
<td>Chevy</td>
<td>92</td>
<td>0</td>
<td>92</td>
</tr>
<tr>
<td>ALL</td>
<td>163</td>
<td>15</td>
<td>178</td>
</tr>
<tr>
<td>Ford</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALL</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

\[ Color \times Model + Color + 1 \rightarrow (Color + 1) \times (Model + 1) \]

we get the following matrix (already transposed):

<table>
<thead>
<tr>
<th></th>
<th>Blue</th>
<th>Green</th>
<th>Red</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Chevy</td>
<td>Ford</td>
<td>ALL</td>
<td>Chevy</td>
</tr>
<tr>
<td>Blue</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Green</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Red</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Cube universality — rolling-up

Then

\[
\rho^\circ \cdot \text{cube } M = \begin{array}{c|cc|c}
\text{Blue} & 
\begin{array}{c|c|c}
\text{Chevy} & 87 & 0 \\
\text{Ford} & 99 & 7 \\
\end{array} \\
\hline
\text{Green} & 
\begin{array}{c|c|c}
\text{Chevy} & 0 & 0 \\
\text{Ford} & 64 & 0 \\
\end{array} \\
\hline
\text{Red} & 
\begin{array}{c|c|c}
\text{Chevy} & 5 & 0 \\
\text{Ford} & 0 & 8 \\
\end{array} \\
\hline
\text{ALL} & 255 & 15 & 270 \\
\end{array}
\]

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

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

- Abadir and Magnus (2005) stress on the need for a standardized notation for linear algebra in the field of econometrics and statistics.
- Since (Macedo and Oliveira, 2013) the authors have invested in typing linear algebra in a way that makes it closer to modern typed languages.
- This talk has shown such a typed approach at work with an example — defining and proving properties of the data cube operator.
- This extends previous efforts on applying LA to OLAP (Macedo and Oliveira, 2015)
- Our main aim is to formalize previous work in the field — e.g. by Datta and Thomas (1999) and by Pedersen and Jensen (2001) — in an unified way.
Future work

- We wish to exploit the **parallelism** inherent in linear algebra (LA) processing to implement data cubing in an efficient, parallel way.
- The properties of **cube** can be used to **optimize** LA scripts involving data cubes.
- Preliminary results (Oliveira, 2016; Pontes et al., 2017) show LA scripts encoding data analysis operations performing better on HPC architectures than standard competitors.
Preliminary results (TPC-H on Search6)

(Filipe Oliveira, Sérgio Caldas, MSc project on HPC)
References


J.N. Oliveira. Towards a linear algebra semantics for query languages, June 2016. Presented at IFIP WG 2.1 #74 Meeting.
U. Strathclyde, Glasgow, 13-17 June (slides available from the WG’s website).

