Nested Data Cubes for OLAP

Stijn Dekeyser,[1] Bart Kuijpers,[⋆1] Jan Paredaens[1], and Jef Wijsen[⋆⋆1]

University of Antwerp (UIA), Dept. Math. & Computer Sci.,
Universiteitsplein 1, B-2610 Antwerp, Belgium
Email: {dekeyser,kuijpers,pareda,jwijsen}@uia.ua.ac.be

Abstract. We present a new model for OLAP, called the *nested data cube* (NDC) model. *Nested data cubes* are a generalization of other OLAP models such as f-tables [3], and hypercubes [2], but also of classical structures such as sets, bags, and relations. The model we propose adds to the previous models mainly flexibility in viewing the data, in that it allows for the assignment of priorities to the different dimensions of the multidimensional OLAP data.
We also present an algebra in which all typical OLAP analysis and navigation operations can be formulated. We present a number of algebraic operators that work on nested data cubes and that preserve the functional dependency between the dimensional coordinates of the data cube and the factual data in it. These operations include nesting, unnesting, summary, roll-up, and aggregation operations. We show how these operations can be applied to sub-NDC's at any depth, and also show that the NDC algebra can express the SPJR algebra [1] of the relational model. A major motivation for defining an algebra rather than a calculus, is that an algebra naturally leads to an implementation strategy. Importantly, we show that the NDC algebra primitives can be implemented by linear time algorithms.

1 Introduction

Since the seminal paper of Codd, Codd, and Salley [5] of 1993, *on-line analytical processing* (OLAP) is recognized as a promising approach for the analysis and navigation of data warehouses and multidimensional data [4, **?**,11, 12, 15, 26]. Multidimensional databases typically are large collections of enterprise data which are arranged according to different dimensions (e.g., time, measures, products, geographical regions) to facilitate sophisticated analysis and navigation for decision support in, for instance, marketing. Figure 1 depicts a three-dimensional database, containing sales information of stores. A popular way of representing such information is the "data cube" [8, 16, 26]. Each dimension is assigned to an axis in n-dimensional space, and the numeric values are placed in the corresponding cells of the 'cube'.

⋆ Post-doctoral research fellow of the Fund for Scientific Research of Flanders (FWO-Vlaanderen).

⋆⋆ At the time the presented research was performed, this author was affiliated to the Free University of Brussels (VUB)

Day : `day`	*Item* : `item`	*Store* : `store`	
Jan1	Lego	Navona	$\rightarrow$ 32
Jan1	Lego	Colosseum	$\rightarrow$ 24
Jan1	Scrabble	Navona	$\rightarrow$ 13
Jan1	Scrabble	Colosseum	$\rightarrow$ 14
Jan1	Scrabble	Kinderdroom	$\rightarrow$ 22
Jan2	Lego	Kinderdroom	$\rightarrow$ 2
Jan2	Lego	Colosseum	$\rightarrow$ 21

Fig. 1. A multidimensional database.

The effectiveness of analysis and the ease of navigation is mainly determined by the flexibility that the system offers to rearrange the perspectives on different dimensions and by its ability to efficiently calculate and store summary information. A sales manager might want to look at the sales per store, or he might want to have the total number of items sold in all his stores in a particular month. OLAP systems aim to offer these qualities.

During the past few years, many commercial systems have appeared that satisfactorily offer, through ever more efficient implementations, a wide number of analysis capabilities. A few well-known examples are Arbor Software's *Essbase* [9], IBM's *Intelligent Server* [18], Red Brick's *Red Brick Warehouse* [22], Pilot Software's *Pilot Decision Support Suite* [21], and Oracle's *Sales Analyzer* [23]. Some of these implementations are founded on theoretical results on efficient array manipulation [19, 20, 24].

In more recent years, however, the need for a formal model for OLAP that explicitly incorporates the notion of different and independent views on dimensions and also offers a logical way to compute summary information, has become apparent. Lately, a number of starting points for such models have been proposed. Gyssens et al. [13] have proposed the first theoretical foundation for OLAP systems: the *tabular database* model. They give a complete algebraic language for querying and restructuring two-dimensional tables. Agrawal et al. [2] have introduced a *hypercube* based data model with a number of operations that can be easily inserted into SQL. Cabibbo and Torlone [3] have recently proposed a data model that forms a logical counterpart of multidimensional arrays. Their model is based on dimensions and *f-tables*. Dimensions are partially ordered categories that correspond to different ways of looking at the multidimensional information (see also [8]). F-tables are structures to store factual data functionally dependent on the dimensions (such as the one depicted in Figure 1). They also propose a calculus to query f-tables that supports multidimensional data analysis and aggregation.

In this paper, we generalize the notions of f-tables and hypercube by introducing the *nested data cube model*. Our data model supports a variety of nested versions of f-tables, each of which corresponds to an assignment of priorities to the different dimensions.

The answer to the query *"Give an overview of the stores and their areas together with the global sales of the various items,"* on the data cube of Figure 1, for instance, is depicted in Figure 2 by a nested data cube. This data cube has two dimensions, *Store* and *Area*, at the most outer level of nesting, and one dimension, *Item*, at a deeper level. This table gives a view of the sales of Figure 1 in such a way that the sales per store and per item are clearly visualized (being grouped in bags).

We also present an *algebra* to formulate queries, such as the one above, on nested data cubes. This query language supports all the important OLAP analysis and navigational restructuring operations on data cubes. There are a number of operations whose aim lies purely in rearranging the information in the cube in order to create different and independent views on the data. These include, for instance, nesting and unnesting operations, factual data collection in bags, duplication, extention of the cube with additional dimensions, and renaming. The algebra also contains selection, aggregation, and roll-up operations which are directed towards the analysis of the data.

For instance, the nested cube of Figure 2 was obtained from the cube of Figure 1 by the following series of operations: first nesting on *Store* is performed, resulting in a cube with *Store* as the only dimension at the highest level, and *Item* and *Day* on a deeper level; then, on the inner level, nesting is now used on *Item*, yielding a nested data cube with depth three, and one dimension in each level of nesting; next, on the deepest level, the information per *Item* is collected resulting in a decrease of the depth by one, the removal of information about *Day*, and the creation of bags of numbers in the functional part of the data cube at depth two; finally, the nested data cube is extended by one dimension in which the roll-up of the *Store* to *Area* information is stored. This results in a nested data cube in which the dimensions *Store* and *Area* are given higher importance than *Item*, and in which *Day* has disappeared. If one is interested in the total number of sales per *Store* and per *Item*, then an aggregate function `sum` can be applied to the individual bags in the cube.

These and other operations are illustrated fully in Section 3, which contains a similar, yet more extensive example.

Motivation. The model we propose offers a natural paradigm for perceiving large data cubes, as it adds to previously mentioned models mainly flexibility in viewing the data by assigning priorities to different dimensions. Put another way, grouping on values of an attribute is made explicit. As is shown in the next section, the NDC model can also be used to represent common data structures such as sets, bags, and relations. Furthermore, our model generalizes and improves upon a number of other OLAP models, as is discussed later in this paragraph.

Let us first come back to the problem of perceiving large data cubes. Traditional data cubes are "flat" in that they treat each dimension in the same way. This causes two kinds of perceptional difficulties: firstly the *dimensionality* can be too high to be practically visualizable in a "cubic" format. Secondly, the *cardinality* (the number of tuples in the database) is typically very high. Our approach can be used to decrease both measures.

We now turn to the comparison of our approach to other OLAP models. While the query language for the tabular data model proposed by Gyssens et al. [13] only covers restructuring of tables, ours supports both restructuring and

Store : `store`	*Area* : `area`		*Item* : `item`	
Navona	Italy	$\rightarrow$	Lego	$\rightarrow \{\!\mid 32 \mid\!\}$
			Scrabble	$\rightarrow \{\!\mid 13 \mid\!\}$
Colosseum	Italy	$\rightarrow$	Lego	$\rightarrow \{\!\mid 24, 21 \mid\!\}$
			Scrabble	$\rightarrow \{\!\mid 14 \mid\!\}$
Kinderdroom	Belgium	$\rightarrow$	Lego	$\rightarrow \{\!\mid 2 \mid\!\}$
			Scrabble	$\rightarrow \{\!\mid 22 \mid\!\}$

Fig. 2. A three-dimensional data cube containing sales information.

complex analysis queries. Their model, although generalized for an arbitrary number of dimensions [14], is mainly suited for two-dimensional spreadsheet applications. The NDC model, however, offers a theoretical framework for "cube viewers" where users navigate through a space of linearly nested n-dimensional cubes.

Our model is based on a simple hypercube model such as the one proposed by Agrawal et al. [2]. Their approach is mainly toward the insertion of their alebra into SQL, while this paper proposes an independent implementation of nested data cubes.

A final comparison of our model with other OLAP models involves the f-tables proposed by Cabibbo et al. [3]. Like their model, ours contains explicit notions of dimensions and describes hierarchies of levels within dimensions in a clean way. However, the model we propose also allows the construction of a hierarchy of the dimensions in an NDC. This does not only make viewing very large cubes easier, it also gives semantics to the scheme of a data cube. Different end-users will typically prefer different schemes for the same underlying data.

An important problem with the calculus for f-tables as proposed by Cabibbo et al. is that in at least two cases it is necessary to leave the model temporarily. Firstly, when aggregate functions are used, the result of a query may no longer be functional. In contrast, the NDC model has the ability to collect factual data in bags which allows us to stay within our model after grouping and before aggregation. Once data is collected in bags, a wide variety of aggregate functions can be performed on them by reusing the constructed bags.

Secondly, in the f-table model, it is not clear where in the system the information for the roll-up function is to be found. It is assumed that it is known for every value in any level how to roll-up to a value in a higher level from the hierarchy. Conversely, our operator that is used for roll-up takes any relation as input, meaning that roll-up information can be stored in another NDC in the system.

While our model clearly shares similarities with the the nested relational model [10, 17, 25], it is not a generalization of it. Importantly, our model imposes linear nesting which only allows for the construction of a linear hierarchy of dimensions.

Organization. Section 2 introduces *nested data cubes* (NDC's). Section 3 introduces the operators of the NDC algebra and illustrates them by presenting an extensive example. Section 4 contains two results concerning the expressive power of the NDC algebra. Section 5 shows how our algebra can be implemented efficiently. Section 6 briefly summarizes the most important results. The appendix contains the formal definitions of the NDC algebra operators. For formal definitions and the proofs of the theorems, we refer to [7].

2 Nested Data Cubes

In this section, we formally define the nested data cube model and illustrate the definitions using the data cube of Figure 2. After giving additional examples, we show that the set of NDC's over a given scheme is recursively enumerable. Algebraic operators that work on NDC's are presented in the next section.

In what follows, we use the delimiters $\{\!|\cdot|\!\}$ to denote a bag. We assume the existence of a set $\mathcal{A}$ of *attributes* and a set $\mathcal{L}$ of *levels*. In Figure 2, the attributes are *Store*, *Area*, and *Item*, and the levels are `store`, `area`, `item`, and `num` (the set of natural numbers). Every level l has a recursively enumerable set $dom(l)$ of atomic values associated to it. For technical reasons, the set $\mathcal{L}$ contains a reserved level, λ, which has a singleton domain: $dom(\lambda) = \{\top\}$, with $\top$ the Boolean true value. We define $\mathbf{dom} = \bigcup\{dom(l) \mid l \in \mathcal{L}\}$.

For certain pairs (l_1, l_2) of $\mathcal{L} \times \mathcal{L}$, there exist a *roll-up function*, denoted $\texttt{R-UP}_{l_1}^{l_2}$, that maps every element of l_1 to an element of l_2. Further requirements may be imposed on the nature of roll-up functions, as is done in [3].

Definition 1. (Coordinate).

- A *coordinate type* is a set $\{A_1 : l_1, \ldots, A_n : l_n\}$ where $A_1, \ldots, A_n$ are distinct attributes, $l_1, \ldots, l_n$ are levels, and $n \geq 0$.
- A *coordinate* over the coordinate type $\{A_1 : l_1, \ldots, A_n : l_n\}$ is a set $\{A_1 : v_1, \ldots, A_n : v_n\}$ where $v_i \in dom(l_i)$, for $1 \leq i \leq n$.

The set of attributes appearing in a coordinate (type) γ, is denoted $att(\gamma)$. □

The NDC of Figure 2 has two coordinate types; i.e., $\{Store : \texttt{store}, Area : \texttt{area}\}$ and $\{Item : \texttt{item}\}$. An example of a coordinate over the former coordinate type is (Navona, Italy).

Definition 2. (Scheme). The abstract syntax of a *nested data cube scheme* (NDC *scheme*, or simply *scheme*) is given by:

$$\tau = [\delta \to \tau] \quad | \quad \beta \qquad (1)$$

$$\beta = l \quad | \quad \{\!|\beta|\!\} \qquad (2)$$

where δ is a coordinate type, and l is a level. Throughout this paper, the Greek characters δ, τ, and β consistently refer to the above syntax. □

The nested data cube of Figure 2, for example, is

$$\tau_0 = [\{\mathit{Store} : \texttt{store}, \mathit{Area} : \texttt{area}\} \to [\{\mathit{Item} : \texttt{item}\} \to \{\!|\texttt{num}|\!\}]].$$

As another example, the scheme of Figure 1 is $[\{\mathit{Day} : \texttt{day}, \mathit{Item} : \texttt{item}, \mathit{Store} : \texttt{store}\} \to \texttt{num}]$.

Definition 3. (Instance). To define an instance (*nested data cube*) over a scheme τ, we first define the function $dom(\cdot)$ as follows:

$$\begin{aligned} dom(\delta) &= \text{the set of all coordinates over coordinate type } \delta \\ dom([\delta \to \tau]) &= \{\{v_1 \to w_1, \ldots, v_m \to w_m\} \mid m \geq 0, \text{ and} \\ &\quad\; v_1, \ldots, v_m \text{ are pairwise distinct coordinates of } dom(\delta), \text{ and} \\ &\quad\; w_i \in dom(\tau) \text{ for } 1 \leq i \leq m\} \\ dom(\{\!|\beta|\!\}) &= \{\{\!|v_1, \ldots, v_m|\!\} \mid v_i \in dom(\beta) \text{ for } 1 \leq i \leq m\} \end{aligned}$$

An NDC over the scheme τ is an element of $dom(\tau)$. □

Figure 2 is a representation of an instance of the NDC with scheme τ_0.

Definition 4. (Depth). The *depth* of a scheme τ, denoted $depth(\tau)$, is defined as the number of occurences of δ in the construction of τ by applying rule (1) of Definition 2.

The notion of depth is extended to NDC's in an obvious way: if C is an NDC over the scheme τ, then we say that the depth of C is $depth(\tau)$.

The notion of *subscheme* of a scheme τ at depth n is assumed to be intuitively clear. □

The NDC with scheme tau_0 is of depth 2. The subscheme at depth 2 is $[\{\mathit{Item} : \texttt{item}\} \to \{\!|\texttt{num}|\!\}]$.

We now give some additional examples.

Example 1. Note that $[\{\} \to \texttt{num}]$ is a legal scheme. Its depth is equal to 1. All NDC's over this scheme can be listed as follows (assume $dom(\texttt{num}) = \{1, 2, \ldots\}$): $\{\}$ (the empty NDC), $\{\{\} \to 1\}$, $\{\{\} \to 2\}$, and so on.

Importantly, `num` itself is also a legal scheme. Its depth is equal to 0. The NDC's over `num` (as a scheme) are 1,2, ...

□

Example 2. NDC's can represent several common data structures, as follows.

*Bag***:** An NDC over a scheme of the form $\{\!|\beta|\!\}$.
*Set***:** An NDC over a scheme of the form $[\{A : l\} \rightarrow \lambda]$.
*Relation***:** An NDC over a scheme of the form $[\delta \rightarrow \lambda]$. The NDC called C_0 in Section 3 represents a conventional relation.
F-tables [3]**:** An NDC over a scheme of the form $[\delta \rightarrow l]$.

□

Example 1 shows that the NDC's over the scheme $[\{\} \rightarrow \texttt{num}]$ are recursively enumerable. The following theorem generalizes this result for arbitrary schemes.

Theorem 1. *The set of all* NDC*'s over a given scheme* τ *is recursively enumerable.*

3 The NDC Algebra

The NDC *algebra* consists of the following 8 operators:

- `bagify` This operator decreases the depth of an NDC by replacing each innermost sub-NDC by a bag containing the right-hand values appearing in the sub-NDC;
- `extend` This operator adds an attribute to an NDC. The attribute values of the newly added attribute are computed from the coordinates in the original NDC;
- `nest` *and* `unnest` These operators capture the classical meaning of nesting and unnesting;
- `duplicate` This operator takes an NDC and replaces the right-hand values by the attribute values of some specified attribute;
- `select` *and* `rename` These operators correspond to operators with the same name in the conventional relational algebra;
- `aggregate` This operator replaces each right-hand value w in an NDC by a new value obtained by applying a specified function to w.

Furthermore, the NDC algebra also allows for the use of these operators at arbitrary depths. For instance, the use of `nest` at depth 2, will be denoted by $\texttt{nest}^2$. For more details, we refer to Section 4.1.

These operators are illustrated in the following extensive example, which resembles the one given in the introduction of this paper, but is purely designed to contain all operators (and thus contains redundant steps). Formal definitions of the operators can be found in the appendix.

Before turning to the example, we make the following remarks.

To make the tables smaller in size, we have used abbreviations for the attributes. It should be noted that the size of the tables printed below is big because we chose to show all sub-cubes at once. However, in interactive cube-viewers, sub-cubes will only “open” when clicked upon, thus reducing the size profoundly.

As a last remark, the reader should understand that the definitions of the operators are formed such that the result of an operation always retains functionality.

The query used in the example is

> *Give an overview per (area, country) pair and per item of the amounts of toys sold over all shops and all time, in the area of Europe.*

We start from raw data in a relation C_0 containing information about toy shops. Typically, such tables may contain more than one attribute that can be seen as a measure. In C_0, for instance, both the number of items sold (So) as well as the number of damaged or lost items (Lo) can serve as a measure.

Da : day	It : item	St : store	So : num	Lo : num	Co : country	
Jan1	Lego	Colosseum	35	4	Italy	$\rightarrow \top$
Jan1	Lego	Navona	12	1	Italy	$\rightarrow \top$
Jan1	Lego	Kindertuin	31	6	Belgium	$\rightarrow \top$
Jan1	Lego	Toygarden	31	1	USA	$\rightarrow \top$
Jan1	Scrabble	Atomium	11	2	Belgium	$\rightarrow \top$
Jan1	Scrabble	Colosseum	15	2	Italy	$\rightarrow \top$
Jan1	Scrabble	Funtastic	22	0	Canada	$\rightarrow \top$
Jan1	Scrabble	Kindertuin	19	5	Belgium	$\rightarrow \top$
Jan1	Scrabble	Navona	17	3	Italy	$\rightarrow \top$
Jan2	Lego	Kindertuin	42	5	Belgium	$\rightarrow \top$
Jan2	Lego	Navona	28	7	Italy	$\rightarrow \top$

C_0

Cube C_1 is obtained after selecting one attribute (So) from C_0 to be used as a measure, i.e., $C_1 = \texttt{duplicate}(C_0, So)$. After chosing this measure for the OLAP analysis, a logical next step would be to remove this and all other measures from the coordinate type of the NDC. This projection can be simulated in our NDC algebra by the following three steps; first the measures are put in a seperate sub-cube by using the `nest` operator, then the information is collapsed in bags, and finally computing the aggregate *ssum* over them.

However, to save space, we temporarily leave the measures in the coordinate type. In later steps, they will disappear.

Da : day	It : item	St : store	So : num	Lo : num	Co : country	
Jan1	Lego	Colosseum	35	4	Italy	$\rightarrow 35$
Jan1	Lego	Navona	12	1	Italy	$\rightarrow 12$
Jan1	Lego	Kindertuin	31	6	Belgium	$\rightarrow 31$
Jan1	Lego	Toygarden	31	1	USA	$\rightarrow 31$
Jan1	Scrabble	Atomium	11	2	Belgium	$\rightarrow 11$
Jan1	Scrabble	Colosseum	15	2	Italy	$\rightarrow 15$
Jan1	Scrabble	Funtastic	22	0	Canada	$\rightarrow 22$
Jan1	Scrabble	Kindertuin	19	5	Belgium	$\rightarrow 19$
Jan1	Scrabble	Navona	17	3	Italy	$\rightarrow 17$
Jan2	Lego	Kindertuin	42	5	Belgium	$\rightarrow 42$
Jan2	Lego	Navona	28	7	Italy	$\rightarrow 28$

$C_1 = \texttt{duplicate}(C_0, So)$

In order to satisfy the query, we have to add the `area` attribute to the coordinate type. This is done by applying `extend`, i.e., $C_2 = \texttt{extend}(C_1, Ar, \texttt{area}, \texttt{R-UP}^{\texttt{area}}_{\texttt{country}})$.

Da : `day`	It : `item`	St : `store`	So : `num`	Lo : `num`	Co : `country`	Ar : `area`	
Jan1	Lego	Colosseum	35	4	Italy	Europe	$\to$ 35
Jan1	Lego	Navona	12	1	Italy	Europe	$\to$ 12
Jan1	Lego	Kindertuin	31	6	Belgium	Europe	$\to$ 31
Jan1	Lego	Toygarden	31	1	USA	America	$\to$ 31
Jan1	Scrabble	Atomium	11	2	Belgium	Europe	$\to$ 11
Jan1	Scrabble	Colosseum	15	2	Italy	Europe	$\to$ 15
Jan1	Scrabble	Funtastic	22	0	Canada	America	$\to$ 22
Jan1	Scrabble	Kindertuin	19	5	Belgium	Europe	$\to$ 19
Jan1	Scrabble	Navona	17	3	Italy	Europe	$\to$ 17
Jan2	Lego	Kindertuin	42	5	Belgium	Europe	$\to$ 42
Jan2	Lego	Navona	28	7	Italy	Europe	$\to$ 28

$C_2 = \texttt{extend}(C_1, Ar, \texttt{area}, \texttt{R-UP}^{\texttt{area}}_{\texttt{country}})$

We now nest in two steps. Thus, $C_3 = \texttt{nest}(C_2, Ar)$.

<table>
<tr><th colspan="3">Ar : <code>area</code></th></tr>
<tr><td>Europe</td><td>$\to$</td><td>
<table>
<tr><th>Da : <code>day</code></th><th>It : <code>item</code></th><th>St : <code>store</code></th><th>So : <code>num</code></th><th>Lo : <code>num</code></th><th>Co : <code>country</code></th><th></th></tr>
<tr><td>Jan1</td><td>Lego</td><td>Colosseum</td><td>35</td><td>4</td><td>Italy</td><td>$\to$ 35</td></tr>
<tr><td>Jan1</td><td>Lego</td><td>Navona</td><td>12</td><td>1</td><td>Italy</td><td>$\to$ 12</td></tr>
<tr><td>Jan1</td><td>Lego</td><td>Kindertuin</td><td>31</td><td>6</td><td>Belgium</td><td>$\to$ 31</td></tr>
<tr><td>Jan1</td><td>Scrabble</td><td>Atomium</td><td>11</td><td>2</td><td>Belgium</td><td>$\to$ 11</td></tr>
<tr><td>Jan1</td><td>Scrabble</td><td>Colosseum</td><td>15</td><td>2</td><td>Italy</td><td>$\to$ 15</td></tr>
<tr><td>Jan1</td><td>Scrabble</td><td>Kindertuin</td><td>19</td><td>5</td><td>Belgium</td><td>$\to$ 19</td></tr>
<tr><td>Jan1</td><td>Scrabble</td><td>Navona</td><td>17</td><td>3</td><td>Italy</td><td>$\to$ 17</td></tr>
<tr><td>Jan2</td><td>Lego</td><td>Kindertuin</td><td>42</td><td>5</td><td>Belgium</td><td>$\to$ 42</td></tr>
<tr><td>Jan2</td><td>Lego</td><td>Navona</td><td>28</td><td>7</td><td>Italy</td><td>$\to$ 28</td></tr>
</table>
</td></tr>
<tr><td>America</td><td>$\to$</td><td>
<table>
<tr><th>Da : <code>day</code></th><th>It : <code>item</code></th><th>St : <code>store</code></th><th>So : <code>num</code></th><th>Lo : <code>num</code></th><th>Co : <code>country</code></th><th></th></tr>
<tr><td>Jan1</td><td>Lego</td><td>Toygarden</td><td>31</td><td>1</td><td>USA</td><td>$\to$ 31</td></tr>
<tr><td>Jan1</td><td>Scrabble</td><td>Funtastic</td><td>22</td><td>0</td><td>Canada</td><td>$\to$ 22</td></tr>
</table>
</td></tr>
</table>

Ar : `area`	Co : `country`	Da : `day`	It : `item`	St : `store`	So : `num`	Lo : `num`	
Europe →	Belgium →	Jan1	Lego	Kindertuin	31	6	→ 31
		Jan1	Scrabble	Atomium	11	2	→ 11
		Jan1	Scrabble	Kindertuin	19	5	→ 19
		Jan2	Lego	Kindertuin	42	5	→ 42
	Italy →	Jan1	Lego	Colosseum	35	4	→ 35
		Jan1	Lego	Navona	12	1	→ 12
		Jan1	Scrabble	Colosseum	15	2	→ 15
		Jan1	Scrabble	Navona	17	3	→ 17
		Jan2	Lego	Navona	28	7	→ 28
America →	USA →	Jan1	Lego	Toygarden	31	1	→ 31
	Canada →	Jan1	Scrabble	Funtastic	22	0	→ 22

$C_4 = \texttt{nest}^2(C_3, Co)$

We now decrease the number of tuples in the cube by performing a selection. Cube $C_5 = \texttt{select}(C_4, Ar,\ \text{Europe})$.

Ar : `area`	Co : `country`	Da : `day`	It : `item`	St : `store`	So : `num`	Lo : `num`	
Europe →	Belgium →	Jan1	Lego	Kindertuin	31	6	→ 31
		Jan1	Scrabble	Atomium	11	2	→ 11
		Jan1	Scrabble	Kindertuin	19	5	→ 19
		Jan2	Lego	Kindertuin	42	5	→ 42
	Italy →	Jan1	Lego	Colosseum	35	4	→ 35
		Jan1	Lego	Navona	12	1	→ 12
		Jan1	Scrabble	Colosseum	15	2	→ 15
		Jan1	Scrabble	Navona	17	3	→ 17
		Jan2	Lego	Navona	28	7	→ 28

$C_5 = \texttt{select}(C_4, Ar,\ \text{Europe})$

To obtain the pairs of (area, country) requested in the query, unnesting is applied to the cube. Cube $C_6 = \texttt{unnest}(C_5)$. Alternatively, we could also have nested the cube C_2 directly in the requested form and applied the selection to the resulting cube.

<table>
<tr><th>Ar : area</th><th>Co : country</th><th></th><th></th></tr>
<tr><td>Europe</td><td>Belgium</td><td>$\rightarrow$</td><td>
<table>
<tr><th>Da : day</th><th>It : item</th><th>St : store</th><th>So : num</th><th>Lo : num</th><th></th></tr>
<tr><td>Jan1</td><td>Lego</td><td>Kindertuin</td><td>31</td><td>6</td><td>$\rightarrow$ 31</td></tr>
<tr><td>Jan1</td><td>Scrabble</td><td>Atomium</td><td>11</td><td>2</td><td>$\rightarrow$ 11</td></tr>
<tr><td>Jan1</td><td>Scrabble</td><td>Kindertuin</td><td>19</td><td>5</td><td>$\rightarrow$ 19</td></tr>
<tr><td>Jan2</td><td>Lego</td><td>Kindertuin</td><td>42</td><td>5</td><td>$\rightarrow$ 42</td></tr>
</table>
</td></tr>
<tr><td>Europe</td><td>Italy</td><td>$\rightarrow$</td><td>
<table>
<tr><th>Da : day</th><th>It : item</th><th>St : store</th><th>So : num</th><th>Lo : num</th><th></th></tr>
<tr><td>Jan1</td><td>Lego</td><td>Colosseum</td><td>35</td><td>4</td><td>$\rightarrow$ 35</td></tr>
<tr><td>Jan1</td><td>Lego</td><td>Navona</td><td>12</td><td>1</td><td>$\rightarrow$ 12</td></tr>
<tr><td>Jan1</td><td>Scrabble</td><td>Colosseum</td><td>15</td><td>2</td><td>$\rightarrow$ 15</td></tr>
<tr><td>Jan1</td><td>Scrabble</td><td>Navona</td><td>17</td><td>3</td><td>$\rightarrow$ 17</td></tr>
<tr><td>Jan2</td><td>Lego</td><td>Navona</td><td>28</td><td>7</td><td>$\rightarrow$ 28</td></tr>
</table>
</td></tr>
</table>

$C_6 = \texttt{unnest}(C_5)$

We now need to nest on the second level of grouping (as mentioned in the statement of our example query), i.e., on the `item` attribute.

<table>
<tr><th>Ar : area</th><th>Co : country</th><th></th><th></th></tr>
<tr><td>Europe</td><td>Belgium</td><td>$\rightarrow$</td><td>
<table>
<tr><th>It : item</th><th></th><th></th></tr>
<tr><td>Lego</td><td>$\rightarrow$</td><td>
<table>
<tr><th>Da : day</th><th>St : store</th><th>So : num</th><th>Lo : num</th><th></th></tr>
<tr><td>Jan1</td><td>Kindertuin</td><td>31</td><td>6</td><td>$\rightarrow$ 31</td></tr>
<tr><td>Jan2</td><td>Kindertuin</td><td>42</td><td>5</td><td>$\rightarrow$ 42</td></tr>
</table>
</td></tr>
<tr><td>Scrabble</td><td>$\rightarrow$</td><td>
<table>
<tr><th>Da : day</th><th>St : store</th><th>So : num</th><th>Lo : num</th><th></th></tr>
<tr><td>Feb1</td><td>Kindertuin</td><td>19</td><td>5</td><td>$\rightarrow$ 19</td></tr>
<tr><td>Feb2</td><td>Atomium</td><td>11</td><td>2</td><td>$\rightarrow$ 11</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>Europe</td><td>Italy</td><td>$\rightarrow$</td><td>
<table>
<tr><th>It : item</th><th></th><th></th></tr>
<tr><td>Lego</td><td>$\rightarrow$</td><td>
<table>
<tr><th>Da : day</th><th>St : store</th><th>So : num</th><th>Lo : num</th><th></th></tr>
<tr><td>Jan1</td><td>Colosseum</td><td>35</td><td>4</td><td>$\rightarrow$ 35</td></tr>
<tr><td>Jan1</td><td>Navona</td><td>12</td><td>1</td><td>$\rightarrow$ 12</td></tr>
<tr><td>Jan2</td><td>Navona</td><td>28</td><td>7</td><td>$\rightarrow$ 28</td></tr>
</table>
</td></tr>
<tr><td>Spider</td><td>$\rightarrow$</td><td>
<table>
<tr><th>Da : day</th><th>St : store</th><th>So : num</th><th>Lo : num</th><th></th></tr>
<tr><td>Jan1</td><td>Colosseum</td><td>15</td><td>2</td><td>$\rightarrow$ 15</td></tr>
<tr><td>Jan1</td><td>Navona</td><td>17</td><td>3</td><td>$\rightarrow$ 17</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

$C_7 = \texttt{nest}^2(C_6, It)$

Since we have obtained the necessary grouping, all attributes remaining at the deepest level of nesting are not needed anymore. Their data is collapsed into bags, i.e., $C_8 = \texttt{bagify}(C_7)$. Note that we are now removing the measures from the coordinate type, and are also putting all information over all dates together.

<table>
<tr><th>Ar : area</th><th>Co : country</th><th></th><th></th></tr>
<tr><td>Europe</td><td>Belgium</td><td>$\rightarrow$</td><td>
<table>
<tr><th>It : item</th><th></th></tr>
<tr><td>Lego</td><td>$\rightarrow \{\!|31, 42|\!\}$</td></tr>
<tr><td>Scrabble</td><td>$\rightarrow \{\!|19, 11|\!\}$</td></tr>
</table>
</td></tr>
<tr><td>Europe</td><td>Italy</td><td>$\rightarrow$</td><td>
<table>
<tr><th>It : item</th><th></th></tr>
<tr><td>Lego</td><td>$\rightarrow \{\!|35, 12, 28|\!\}$</td></tr>
<tr><td>Scrabble</td><td>$\rightarrow \{\!|15, 17|\!\}$</td></tr>
</table>
</td></tr>
</table>

$$C_8 = \texttt{bagify}(C_7)$$

As a last step in the implementation of our example query in the NDC algebra, we perform the `sum` aggregate on the bags of cube C_8 to obtain the totals of sold items. This yields the desired result.

Ar : `area`	Co : `country`		It : `item`		
Europe	Belgium	→	Lego	→	73
			Scrabble	→	30
Europe	Italy	→	Lego	→	75
			Scrabble	→	32

$$C_9 = \texttt{aggregate}(C_8, \texttt{sum})$$

4 The Expressive Power of the NDC Algebra

In this section, we show some properties concerning the expressiveness of the NDC algebra. We first show that the NDC algebra is sufficiently powerful to capture algebraic operations working directly on sub-NDC's over a subscheme of a scheme. Next we show that the NDC algebra can express the SPJR algebra [1].

We refer to [7] for the proofs of the theorems in this section.

4.1 Applying Operators at a Certain Depth

The recursion in the definition of NDC is a "tail recursion." Consequently, the "recursion depth" can be used to unequivocally address a sub-NDC within an NDC. This is an interesting and important property of NDC's. It is exploited by defining operators that directly work on sub-NDC's at a certain depth. Such operators reduce the need for frequent nesting and unnesting of NDC's.

Definition 5. Let C be an NDC over the scheme τ. Let $1 \leq d \leq \mathit{depth}(\tau)$. Let $\texttt{op}(C, a_1, \ldots, a_n)$ be any previously defined operation of the NDC algebra.

Let $\tau' = \mathit{subscheme}(\tau, d)$. $\texttt{op}^d(C, a_1, \ldots, a_n)$ is defined iff $\texttt{op}(\tau', a_1, \ldots, a_n)$ is defined.

We first give the result on schemes.

1. $\texttt{op}^1(\tau, a_1, \ldots, a_n) = \texttt{op}(\tau, a_1, \ldots, a_n)$.
2. If $d > 1$ then $\texttt{op}^d([\delta \to \tau], a_1, \ldots, a_n) = [\delta \to \texttt{op}^{d-1}(\tau, a_1, \ldots, a_n)]$.

We next give the result on NDC's.

1. $\texttt{op}^1(C, a_1, \ldots, a_n) = \texttt{op}(C, a_1, \ldots, a_n)$.
2. If $d > 1$ and $C = \{v_1 \to w_1, \ldots, v_m \to w_m\}$ then $\texttt{op}^d(C, a_1, \ldots, a_n) = \{v_1 \to \texttt{op}^{d-1}(w_1, a_1, \ldots, a_n), \ldots, v_m \to \texttt{op}^{d-1}(w_m, a_1, \ldots, a_n)\}$.

□

In the example of Section 3, cube C_4 was obtained from C_3 by using the `nest` operator at depth 2.

The following theorem states that $\mathtt{op}^d(C, a_1, \ldots, a_n)$ is not a primitive operator—i.e., it can be expressed in terms of the operators of the NDC algebra.

Theorem 2. *Let* C *be an* NDC *over the scheme* τ*. The operator* $\mathtt{op}^d(C, a_1, \ldots, a_n)$ *with* $d \geq 2$ *is redundant.*

As an example of this theorem, cube C_4 of the previous section can be obtained from cube C_3 by applying the following expressions at depth 1:

$$C_4 = \mathtt{nest}(\mathtt{nest}(\mathtt{unnest}(C_3), It), Ar).$$

4.2 The SPJR Algebra

The following theorem states that the NDC algebra can express the SPJR algebra [1].

Theorem 3. *The* NDC *algebra expresses the SPJR algebra.*

The proof for Theorem 3 (see [7]) shows how the `extend` operator can be used to simulate the relational join.

5 Implementing the NDC Algebra

The operations of Section 3 can be implemented by algorithms that run in linear time with respect to the number of atomic values that appear in the data cube. We assume that each aggregate function is computable in polynomial time.

We introduce two new constructs: the *i*Cube which holds the actual data in an n-dimensional array, and the *i*Struct, essentially a string representing the structure behind the data.

For example, consider the scheme $[\{A_1 : \mathtt{a}_1, A_2 : \mathtt{a}_2\} \to [\{B_1 : \mathtt{b}_1, B_2 : \mathtt{b}_2\} \to \mathtt{c}]]$. It can be implemented by the *i*Cube *cube* of type $\mathtt{c}[\#\mathtt{b}_1][\#\mathtt{a}_2][\#\mathtt{d}_1][\#\mathtt{b}_2][\#\mathtt{a}_1]$ (the array type of the Java language is used for simplicity) together with the *i*Struct $[5, 2 \to [1, 4 \to \cdot]]$. In the *i*Cube's type, $\#\mathtt{a}_1$ denotes the cardinality of $dom(\mathtt{a}_1)$ plus one. That is, there is one entry for each element of $dom(\mathtt{a}_1)$ (indexes $1, 2, \ldots, \#\mathtt{a}_1 - 1$) on top of the entry with index 0. The numbers in the *i*Struct denote positions in the array type. For example, "5" refers to the fifth dimension, which ranges to $\#\mathtt{a}_1$. A possible member of an NDC over the given scheme is $[\{A_1 : u_1, A_2 : u_2\} \to [\{B_1 : v_1, B_2 : v_2\} \to w]]$ which will be represented in the *i*Cube as $cube[v_1][u_2][0][v_2][u_1] = w$.

Note that an extra, unused dimension is present in the *i*Cube (namely, the third dimension ranging to $\#\mathtt{d}_1$). This is necessary in case the `extend` operation is used, as we require the *i*Cubes to remain the same throughout the computation of the query. This dimension will then be used to store the attribute created by the `extend` operator.

A final remark relates to the use of bags in our model. The implementation should support fast access to the elements in a bag of an NDC, to facilitate the computation of aggregate functions. Consider, for example, the scheme $[\{A : \mathtt{a}\} \to \{\!|\mathtt{c}|\!\}]$. NDC's over this scheme contain bags. One possible implementation uses the *i*Cube *cube* of type $\mathtt{c}[\#\mathtt{a}][\#\mathtt{d}]$ and the *i*Struct $[1 \to \{\!|2|\!\}]$. A member $[\{A : v\} \to \{\!|w_1, w_2|\!\}]$ of an NDC over this scheme, for example, is represented by $\{\!|\mathit{cube}[v][i] \mid 1 \leq i < \#\mathtt{d}|\!\} = \{\!|w_1, w_2|\!\}$.

The operations of the NDC algebra are implemented in such a way that the type of the *i*Cube never changes. Importantly, the `nest`, `unnest`, and `bagify` operations only change the *i*Struct, leaving the *i*Cube unaffected. The other operations also change the content of the *i*Cube. Based on this, we can implement an expression in the NDC algebra in linear time. We now give a concrete example.

Let *revenue* be an NDC over the scheme

$$[\{\mathit{Store} : \mathtt{store} \to [\{\mathit{City} : \mathtt{city}\} \to [\{\mathit{Product} : \mathtt{product}\} \to \mathtt{num}]]]$$

We want to answer the query *"For each city, give the maximal total revenue realized by any store in that city."* The *i*Cube and initial *i*Struct implementing the NDC are $\mathit{revenue} = \mathtt{num}[\#\mathtt{store}][\#\mathtt{city}][\#\mathtt{product}]$ and $[1 \to [2 \to [3 \to \cdot]]]$, respectively. The algebraic expression for the query is

$$\mathtt{aggregate}(\mathtt{bagify}(\mathtt{aggregate}(\mathtt{bagify}(\mathtt{nest}(\mathtt{unnest}(\mathit{revenue}), \mathit{City})), \mathtt{sum})), \mathtt{max})$$

A Java-like linear program implementing the expression is

```
for (int c = 1; c ≤ #city; c++) {
   revenue[0][c][0] = 0;
   for (int s = 1; s ≤ #store; s++) {
      revenue[s][c][0] = 0;
      for (int p = 1; p ≤ #product; p++) {
         revenue[s][c][0] += revenue[s][c][p];
      }
      revenue[0][c][0] = max(revenue[0][c][0], revenue[s][c][0]);
   }
}
```

6 Summary

We proposed the NDC data model and its associated algebra. The NDC data model differs from most existing OLAP data models in its explicit modeling of grouping at different levels of nesting. We believe that nested grouping naturally arises in many OLAP applications. We proved some properties about the expressiveness of our algebra. The advantage of an algebra in comparison with a calculus is that an algebra approach is generally more close to an implementation. We indicated, in fact, that all operations of the NDC algebra can be implemented by linear time algorithms.

References

1. S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.
2. R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *Proc. IEEE Int. Conf. Data Engineering (ICDE '97)*, pages 232–243, 1997.
3. L. Cabibbo and R. Torlone. Querying multidimensional databases. In *Sixth Int. Workshop on Database Programming Languages (DBPL '97)*, pages 253–269, 1997.
4. S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65–74, 1997.
5. E. Codd, S. Codd, and C. Salley. Providing OLAP (On-Line Analytical Processing) to user-analysts: An IT mandate. *Arbor Software White Paper,* `http://www.arborsoft.com`.
6. G. Colliat. Olap, relational, and multidimensional database systems. *SIGMOD Record*, 25(3):64–69, 1996.
7. S. Dekeyser, B. Kuijpers, J. Paredaens, and J. Wijsen. Nested Data Cubes. *Technical Report 9804*, University of Antwerp, 1998. `ftp://wins.uia.ac.be/pub/olap/ndc.ps`
8. C. Dyreson. Information retrieval from an incomplete data cube. In *Proc. Int. Conf. Very Large Data Bases (VLDB '96)*, pages 532–543, Bombai, India, 1996.
9. Essbase. *Arbor Software,* `http://www.arborsoft.com/OLAP.html`.
10. P.C. Fischer, and S.J. Thomas. Nested Relational Structures. In *The Theory of Databases, Advances in Computing Research III*, PC. Kanellakis, ed., pages 269–307, JAI Press, Greenwich, CT, 1986.
11. J. Gray, A. Boswirth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by. In *Proc. IEEE Int. Conf. Data Engineering (ICDE '97)*, pages 152–159, 1997.
12. J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *Data Mining and Knowledge Discovery*, 1:29–53, 1997.
13. M. Gyssens, L. Lakshmanan, and I. Subramanian. Tables as a paradigm for querying and restructuring. In *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '96)*, pages 93–103, Montreal, Canada, 1996.
14. M. Gyssens and L. Lakshmanan. A Foundtion for Multi-Dimensional Databases. In *Proc. Int. Conf. Very Large Data Bases (VLDB '97)*, pages 106–115, Athens, Greece, 1997.
15. J. Han. OLAP mining: An integration of OLAP with data mining. In *Proceedings of the 7th IFIP 2.6 Working Conference on Database Semantics (DS-7)*, pages 1–9, 1997.
16. V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *Proc. ACM SIGMOD International Conference on Management of Data (SIGMOD '96)*, pages 205–216, Montreal, Canada, 1996.

17. G. Jaeschke, and H.-J. Schek. Remarks on the Algebra on Non First Normal Form Relations. In *Proceedings first Symposium on Principles of Database Systems (PODS '82)*, pages 124–138, Los Angeles, CA, 1982.
18. Intelligent server. *IBM,* `http://www.software.ibm.com/data/pubs/papers`.
19. L. Libkin, R. Machlin, and L. Wong. A query language for multidimensional arrays: Design implementation, and optimization techniques. In *Proc. ACM SIGMOD Int. Conf. Management of Data (SIGMOD '96)*, pages 228–239, Montreal, Canada, 1996.
20. A. Marathe and K. Salem. A language for manipulating arrays. In *Proc. Int. Conf. Very Large Data Bases (VLDB '97)*, pages 46–55, Athens, Greece, 1997.
21. Pilot decision support suite. *Pilot Software*, `http://rickover.pilotsw.com/products/Welcome.htm`.
22. Red brick warehouse. *Red Brick*, `http://www.redbrick.com/rbs-g/html/plo.html`.
23. Sales analyzer. *Oracle*, `http://www.oracle.com/products/olap/html/`.
24. S. Sarawagi and M. Stonebraker. Efficient organization of large multidimensional arrays. In *Proc. IEEE Int. Conf. Data Engineering (ICDE '94)*, pages 328–336, Houston, Texas, 1994.
25. H. J. Schek, and M. H. Scholl. The Relational Model with Relation-Valued Attributes. In *Information Systems* **11:2**, pages 137–147, 1986.
26. A. Shoshani. OLAP and statistical databases: Similarities and differences. In *Proc. ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '97)*, pages 185–196, Tucson, AZ, 1997.

Appendix

In the appendix, we formally define the operators that were introduced in Section 3. Operators of the NDC algebra are defined on scheme level and on instance (NDC) level.

The `bagify` operator

The $\texttt{bagify}(\cdot)$ operator takes as its argument an NDC of depth $n \geq 1$, and returns a new NDC with depth $(n-1)$. Intuitively, this operator is related to the projection of the relational algebra, as it removes attributes.

Definition 6. Let C be an NDC over the scheme $[\delta \to \tau]$.

On scheme level, $\texttt{bagify}([\delta \to \tau])$ is recursively defined as follows:

$$\texttt{bagify}([\delta_1 \to [\delta_2 \to \tau]]) = [\delta_1 \to \texttt{bagify}([\delta_2 \to \tau])]$$
$$\texttt{bagify}([\delta \to \beta]) = \{\!|\beta|\!\}$$

On NDC's, $\texttt{bagify}(C)$ is recursively defined as follows.

- If $C = \{v_1 \to w_1, \ldots, v_m \to w_m\}$ is an NDC over $[\delta_1 \to [\delta_2 \to \tau]]$, then $\texttt{bagify}(C) = \{v_1 \to \texttt{bagify}(w_1), \ldots, v_m \to \texttt{bagify}(w_m)\}$.
- If $C = \{v_1 \to w_1, \ldots, v_m \to w_m\}$ is an NDC over $[\delta \to \beta]$, then $\texttt{bagify}(C) = \{\!|w_1, \ldots, w_m|\!\}$.

□

The `extend` operator

Definition 7. Let C be an NDC over the scheme $[\delta \to \tau]$. Let A be an attribute such that $A \notin att(\delta)$. Let l be a level. Let R be a subset of $dom(\delta) \times dom(l)$.

The definition of `extend` on scheme level is as follows. $\texttt{extend}([\delta \to \tau], A, l, R)$ is equal to $[\delta \cup \{A : l\} \to \tau]$.

The definition of `extend` on NDC level is as follows. $\texttt{extend}(C, A, l, R)$ is defined as the smallest NDC containing $v \cup \{A : c\} \to w$ whenever (1) C contains $v \to w$, and (2) $(v, c) \in R$. □

As discussed in the motivation, the relation R used in $\texttt{extend}(\cdot, \cdot, \cdot, R)$ can be given by an NDC. This is an interesting feature, as it allows for both the storage of roll-up information, and the "joining" of two NDC's.

Intuitively, it is clear that the relation R can not be represented by an arbitrary NDC; a nested NDC is not possible for instance. For a formal discussion of how NDC's can be used as input relations to the extend operator, we again refer to [7].

The `nest` operator

Definition 8. Let C be an NDC over the scheme $[\delta \to \tau]$, X a subset of $att(\delta)$, and $Y = att(\delta) \setminus X$. Let χ be the coordinate type δ restricted to attributes of X, and ψ the coordinate type δ restricted to attributes of Y.

On scheme level, $\texttt{nest}([\delta \to \tau], X)$ is equal to $[\chi \to [\psi \to \tau]]$.

On instance (NDC) level, $\texttt{nest}(C, X)$ is defined as follows. Let

$$V = \{v[X] \mid v \to w \in C\}.$$

Clearly, $V \subseteq dom(\chi)$. Then

1. For every $x \in V$, $\texttt{nest}(C, X)$ contains $x \to C'$ where C' is the NDC over $[\psi \to \tau]$ satisfying

$$C' = \{v[Y] \to w \mid v \to w \in C, \text{ and } v[X] = x\}.$$

2. If $\texttt{nest}(C, X)$ contains $x \to C'$ then $x \in V$—i.e., $\texttt{nest}(C, X)$ contains no other elements than those specified in (1).

□

The `unnest` operator

The `unnest` operator is the inverse of the `nest` operator.

Definition 9. Let C be an NDC over the scheme $[\delta_1 \to [\delta_2 \to \tau]]$, where $att(\delta_1) \cap att(\delta_2) = \{\}$.

On scheme level, $\texttt{unnest}([\delta_1 \to [\delta_2 \to \tau]])$ is equal to $[\delta_1 \cup \delta_2 \to \tau]$.

On instance level, $\texttt{unnest}(C)$ is the smallest (w.r.t. set inclusion) NDC containing $v_1 \cup v_2 \to w$ whenever C contains $v_1 \to C'$ with $v_2 \to w \in C'$. □

The `duplicate` operator

Informally, `duplicate` serves to duplicate attribute values at the right-hand side in an NDC. Both Codd et al. [5] and Agrawal et al. [2] stress the importance of "symmetric treatment of dimensions and measures", meaning that it needs to be possible to transfer the right-hand values appearing in the deepest sub-cube of an NDC (the *measures*) to the coordinates (the *dimensions*) of that NDC *and* vice versa. The first direction is made possible by the `extend` operator, while `duplicate` facilitates the latter direction.

Definition 10. Let C be an NDC over the scheme $[\delta \to \tau]$, and $A : l \in \delta$.

On scheme level, $\texttt{duplicate}([\delta \to \tau], A)$ is equal to $[\delta \to l]$.

On instance level, $\texttt{duplicate}(C, A)$ is the smallest NDC over $[\delta \to l]$ containing $v \to c$ whenever C contains $v \to C'$ and $v(A) = c$ (for some NDC C' over τ).

□

The `select` operator

Definition 11. Let C be an NDC over the scheme $[\delta \to \tau]$. Let $A : l \in \delta$. Let $B \in att(\delta)$, and $c \in dom(l)$.

For both types of the operator, i.e., $\texttt{select}([\delta \to \tau], A, c)$ and $\texttt{select}([\delta \to \tau], A, B)$, the scheme of the result is equal to $[\delta \to \tau]$.

On instance level, $\texttt{select}(C, A, c)$ is the smallest NDC over $[\delta \to \tau]$ containing $v \to w$ whenever C contains $v \to w$ and $v(A) = c$, while $\texttt{select}(C, A, B)$ is the smallest NDC over $[\delta \to \tau]$ containing $v \to w$ whenever C contains $v \to w$ and $v(A) = v(B)$. □

The `rename` operator

The `rename` operator serves to rename attributes.

Definition 12. Let C be an NDC over the scheme $[\delta \to \tau]$. Let $A \in att(\delta)$ and let B be an attribute not in $att(\delta)$. Let δ' be the coordinate type obtained from δ by substituting B for A.

On scheme level, $\texttt{rename}([\delta \to \tau], A, B)$ is equal to $[\delta' \to \tau]$.

On instance level, $\texttt{rename}(C, A, B)$ is the smallest NDC over $[\delta' \to \tau]$ containing $v \cup \{B : c\} \to w$ whenever C contains $v \cup \{A : c\} \to w$. □

The `aggregate` operator

The `aggregate` operator applies a (aggregation) function on the right-hand values appearing in an NDC.

Definition 13. The *ground* of a scheme τ, denoted $ground(\tau)$, is defined recursively as follows:

$$\begin{aligned} ground([\delta \to \tau]) &= ground(\tau) \\ ground(\beta) &= \beta \end{aligned}$$

Let C be an NDC over the scheme τ with $ground(\tau) = \beta_1$. Let f be a total function from $dom(\beta_1)$ to $dom(\beta_2)$.

On scheme level, $\texttt{aggregate}(\tau, f)$ is recursively defined as follows:

$$\begin{aligned} \texttt{aggregate}([\delta \to \tau], f) &= [\delta \to \texttt{aggregate}(\tau, f)] \\ \texttt{aggregate}(\beta_1, f) &= \beta_2 \end{aligned}$$

On NDC's, $\texttt{aggregate}(C, f)$ is recursively defined as follows.

- If $C = \{v_1 \to w_1, \ldots, v_m \to w_m\}$ is an NDC over $[\delta \to \tau]$, then $\texttt{aggregate}(C, f) = \{v_1 \to \texttt{aggregate}(w_1, f), \ldots, v_m \to \texttt{aggregate}(w_m, f)\}$.
- If $C = c$ is an NDC over β_1, then $\texttt{aggregate}(C, f) = f(c)$.

$\square$